1973-12

A study of deterministic survivable networks.

Labre, Ruben F.

Monterey, California. Naval Postgraduate School

A STUDY OF DETERMINISTIC SURVIVABLE NETWORKS

Ruben F. Labre

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A STUDY OF DETERMINISTIC

SURVIVABLE NETWORKS

by

Ruben F. Labre

Thesis Advisor: Charles H. Rothauge

December 1973

Approved for public release; distribution unlimited.

A Study of Deterministic Survivable Networks

by

Ruben F. Labre
Lieutenant, Philippine Navy
B.S., Naval Postgraduate School, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1973

ABSTRACT

The idea of survivability introduced as a network parameter has led to so many investigations. Several measures of survivability has been studied. The number of links and/or stations needed to be damaged to disrupt the system is the survivability criterion adapted in this study.

The development of analysis procedures for directed, undirected, or mixed networks based on the above criterion and use of the concepts in network flow and graph theory are treated in detail including computer program implementation of the algorithms. Finally a practical design algorithm for minimum-cost survivable network with respect to branch disconnection using a heuristic approach and analysis techniques is described.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I wish to express my gratitude to my thesis advisor, Doctor Charles H. Rothauge of the Department of Electrical Engineering, Naval Postgraduate School, for helping me strengthen my background in network flows, and for his continued guidance and encouragement during all the phases of my work.

I. INTRODUCTION

Network complexes such as railroads, highways, communication circuits, telephone systems, and many others which are interconnections of links and stations, and where there exist commodity flow, have been modelled by linear graphs. The vertices of the graph symbolize stations or terminals while the branches denote lines, trunks, or circuits. The graphical structure of such systems allows the use of topological concepts in the development of the theory of network.

In 1961 the idea of survivability as a graph parameter was introduced. This concept was later adapted as a network parameter. Survivability of a network is defined as its capability to function as a system after damage induced by enemy attack. Survivability and vulnerability are used interchangeably by many authors. The term network reliability is more appropriately used when referring to damage caused by natural disturbances.

The study of network survivability employs topological methods and the concept of network flow. There are two distinct areas, namely: deterministic survivability and probabilistic survivability. A network is considered probabilistic when the parameters involved like commodity flow and weights assigned to links and stations are statistical. When there is certainty in the existence of these parameters, the network becomes deterministic.

Up to present, network survivability has been the subject of research by many graph and network theoreticians. Recent works were focused on the mathematical formulation of survivability criteria, the development of analysis methods, and the synthesis of optimally invulnerable networks. Numerous results have already been published.

This thesis is based on the paper by Frank and Frisch [10] which summarizes the most significant results of the researches in this area. The treatment of deterministic networks is restated in more detail and extended by adapting other results in network flow theory with some modifications to provide additional analysis procedures. To provide working tools for analysis and synthesis, the existing algorithms and those that were modified were coded for the computers. Further simplifications were introduced when encoding these algorithms into the programs which appear in the appendices attached herewith.

In Chapter II of this report, a summary of the important elementary principles of network theory essential in the study of network survivability is presented. Chapter III covers the analysis procedures based on some survivability criteria. The principles from which they were derived are also discussed. Investigation of directed, undirected, and mixed networks with respect to branch disconnection and vertex disconnection are separately treated.

These analysis tools may be employed in the design of optimally survivable networks. The object of Chapter IV is design optimization by an heuristic approach using analysis techniques.

II. NETWORK FUNDAMENTALS

2.1 NETWORK TOPOLOGY

One of the many applications of linear graphs to engineering problems is the topological representation of networks. Networks may be categorized as "electrical networks", "switching networks", and "flow networks". The latter which is often called "communication net" has associated with its branches some information or commodity flow. Symbols, terminology, and a minimum of the fundamental concepts necessary in the study of this type of network is discussed in this chapter.

2.1.1 Directed, Undirected, and Mixed Networks

A directed network is represented by a directed linear graph $G=(V,\Gamma)$, consisting of a set of elements called vertices denoted by the symbol V and a set of ordered pairs of vertices called branches denoted by Γ. The vertex v_i is called the initial vertex and the vertex v_j the terminal vertex; vertices v_i and v_j are said to be adjacent.

A directed branch is indicated by the ordered pair (i,j). Each branch carries an orientation and the information flow is only in the specified direction.

An undirected network is represented by an undirected graph $\overline{G}=(V,\overline{\Gamma})$, consisting of a set of vertices V, and a set of unordered pairs of these vertices denoted by $\overline{\Gamma}$. An undirected branch is indicated by [i,j] or [j,i]. The flow of information is bidirectional.

A mixed network is symbolized by a graph that has both directed and undirected branches. A non-oriented branch may be represented graphically by two parallel branches oppositely directed.

The network graph may have more than one branch between the same pair of vertices.

2.1.2 Paths In a Network Graph

A directed s-t path π_i is a sequence of distinct vertices and branches connecting two arbitrary vertices, v_s and v_t. It has the form

$$v_s, (s,i_1), v_{i_1}, (i_1,i_2), v_{i_2}, \ldots, v_{i_k}, (i_k,t), v_t$$

An undirected s-t path π_i is a sequence of distinct vertices and branches of the form

$$v_s, [s,i_1], v_{i_1}, [i_1,i_2], v_{i_2}, \ldots, v_{i_k}, [i_k,t], v_t$$

Two directed or undirected paths π_i and π_j are branch disjoint if they have no branches in common and vertex disjoint if the only vertices they have in common are v_s and v_t.

In a sequence of vertices and branches describing a directed path, if (i,j) is in the path, then (i,j) is a forward branch with respect to the path. A branch (j,i) in the path is a backward branch with respect to the path.

The network graph G is said to be connected if for any pair of vertices v_i and v_j there is a path π_{i-j} in G. Otherwise it is disconnected. The resulting subgraphs are components.

2.1.3 Cutsets and Cuts

An undirected branch cutset of an undirected, connected graph is the minimal set of branches, the removal of which yields a graph of two or more components.

A directed branch cutset is a minimal set of branches of a directed graph, the removal of which disconnects all directed paths from at least one vertex of G to at least another vertex of G.

An undirected vertex cutset of an undirected, connected graph is the minimal set of vertices whose removal from $\bar{G}$ disconnects the graph.

A directed vertex cutset is a set of vertices of a directed graph whose removal from G destroys all directed paths from at least one of the remaining vertex of G to at least another remaining vertex.

Cutsets with respect to a vertex pair v_s and v_t are referred to as s-t cutsets.

A cut $(X,\bar{X})$ separating v_s and v_t is a set of branches (i,j) such that $v_s \in X$ and $v_t \in \bar{X}$. In Fig. 2.1, if $X = \{v_s, v_1, v_2\}$ and $\bar{X} = \{v_3, v_4, v_t\}$, the set of branches $(X,\bar{X}) = \{(1,4), (1,3), (2,3)\}$ is a directed s-t cut. Branch (2,4) is oppositely directed with respect to the vertices s and t therefore it does not belong to the set.

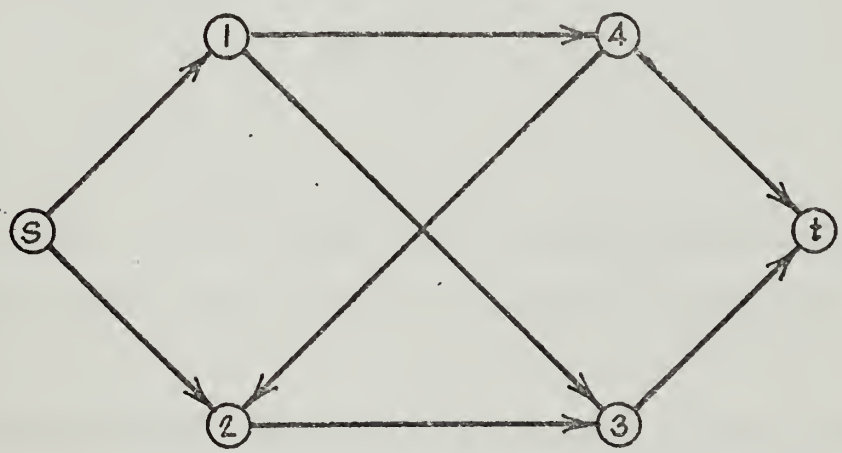


Fig. 2.1

2.1.4 Network Tree

A spanning tree T = (V,U) of a connected network graph G = (V, Γ) is a connected subgraph of G containing all of the vertices of G but no closed path.

If n is the number of vertices in G, a tree contain n-1 branches.

Every pair of vertices v_i and v_j of graph T is joined by one and only one i-j path. Therefore it becomes disconnected when one branch is removed.

2.2 FLOWS IN DETERMINISTIC NETWORK

2.2.1 Branch Flow and Branch Capacity

Aside from the topological structure of a communication net, there is assigned to each branch a finite weight f(i,j) or f[i,j] called the branch flow from v_i to v_j. Also a weight c(i,j) or c[i,j] is assigned to each branch which is the maximum amount of commodity that can reach v_j from v_i per unit time, called the branch capacity.

For an undirected network,

$$c[i,j] = -c[j,i], \qquad (2.2.1.1)$$

$$f[i,j] = -f[j,i], \qquad (2.2.1.2)$$

$$f[s,t] = \Sigma\, f[i,j], \qquad (2.2.1.3)$$

where v_s and v_t are any chosen source and sink vertices respectively. For a directed case, equations (2.2.1.1) and (2.2.1.2) are not necessarily true.

An undirected branch can be represented by two oppositely directed branches between v_i and v_j with capacities

$$c(i,j) = c(j,i) = c[i,j]. \qquad (2.2.1.4)$$

2.2.2 Flow Pattern

The set of flows associated with the branches in G is called a flow pattern denoted by F. A flow pattern is said to be feasible if it satisfies Equations (2.2.2.1) and (2.2.2.2) for some nonnegative constant $f_{s,t}$ which is called the value of F.

For all $v_i \in V$ of a directed network,

$$f(i,V) - f(V,i) = \begin{cases} f_{s,t} & \text{if } i=s \\ 0 & \text{if } i \neq s,t \\ -f_{s,t} & \text{if } i=t \end{cases} \qquad (2.2.2.1)$$

$$0 \leq f(i,j) \leq c(i,j) \text{ for all } i \text{ and } j \qquad (2.2.2.2)$$

This means that the net flow out of the source v_s is $f_{s,t}$ and the flow out of the sink v_t is $-f_{s,t}$, whereas the net flow out of an intermediate vertex is zero. This latter relation is also true for an undirected network.

A branch (i,j) is saturated if $f(i,j) = c(i,j)$ and unsaturated otherwise. It is said to be empty if $f(i,j) = 0$.

2.2.3 Residual Capacity and Augmentation Path

The residual capacity $r(\pi_{s,t})$ is defined both for directed and undirected network as

$$r(\pi_{s,t}) = \text{Min} \left[\underset{\substack{\text{backward} \\ \text{branches} \\ (i,j) \in \pi_{s,t}}}{\text{Min}} \left[f(i,j) \right] , \underset{\substack{\text{forward} \\ \text{branches} \\ (a,b) \in \pi_{s,t}}}{\text{Min}} \left[c(a,b) - f(a,b) \right] \right] \qquad (2.2.3.1)$$

A path $\pi_{s,t}$ is saturated if $r(\pi_{s,t}) = 0$ and unsaturated otherwise. Additional flow can be sent through an unsaturated path obtaining a new flow pattern. To achieve this, add $r(\pi_{s,t})$ to all forward flows and subtract $r(\pi_{s,t})$ from all backward flows along the path.

If a path between two vertices has

$$f(i,j) < c(i,j)$$

and $f(j,i) > 0,$

then we call this a flow augmentation path. The flow along this path can be increased according to the previously stated rule.

2.2.4 Maximizing Network Flow

One of the most important and basic theorems in network flows is the "Max-flow Min-cut theorem" formulated by Ford and Fulkerson [7]. The problem of obtaining the maximum obtainable flow value $f_{s,t}$ through a net with reference to two vertices v_s and v_t is solved using this theorem.

Max-flow Min-cut Theorem: The maximal flow value obtainable in a network G form v_s to v_t is the minimum of the value of a cutset taken over all cutsets separating v_s and v_t.

To state the theorem mathematically, let:

$\tau_{s,t}$ be the value of the minimum s-t cut

$c(X_i,\bar{X}_i)$ be the capacity of a cut $(X_i,\bar{X}_i)$

$\left\{(X_i,\bar{X}_i)_{s,t}\right\}$ be the set of all s-t cuts in the net

then:

$$\max_F \left[f_{s,t}\right] = \min_i \left[c(X_i,\bar{X}_i)_{s,t}\right] = \tau_{s,t} \qquad (2.2.4.1)$$

The value v of the cut $(X,\bar{X})$ is defined as

$$v = f(X,\bar{X}) - f(\bar{X},X) \leq c(X,\bar{X}) \qquad (2.2.4.2)$$

The Ford and Fulkerson Labeling and Augmentation algorithm [7] based on the above theorem provides a method of increasing $f_{s,t}$ systematically from an arbitrary feasible flow pattern with value less than $\tau_{s,t}$ and terminates when $f_{s,t} = \tau_{s,t}$. The resulting flow pattern is maximal.

The algorithm will be stated in a modified form in Chapter III.

2.3 MATRICES ASSOCIATED WITH SURVIVABLE NETWORKS

Network graphs may be structurally drawn or be described by matrices. The latter representation is useful for computer calculations. This section enumerates several matrices used in the study of survivable networks.

2.3.1 The Incidence Matrix

Let $G = (V, \Gamma)$ be the directed graph of the network consisting of n vertices and m branches. Let $A = [a_{i,j}]$ be an n x m matrix whose i-jth entry is $a_{i,j}$. Then A is the incidence matrix of G if

$$a_{i,j} = \begin{cases} +1 & \text{if a branch is directed away from } v_i \\ -1 & \text{if a branch is directed towards } v_i \\ 0 & \text{otherwise} \end{cases}$$

If G is an undirected graph, $A = [a_{i,j}]$ is the incidence matrix if

$$a_{i,j} = \begin{cases} 1 & \text{if branch is incident at } v_i \\ 0 & \text{otherwise} \end{cases}$$

2.3.2 Connection Matrix

An alternate representation of a non-oriented network is the connection matrix $K = [k_{i,j}]$. K is an n x n symmetric matrix whose i-jth entry is $k_{i,j}$ where

$$k_{i,j} = \begin{cases} 1 & \text{if } [i,j] \in \Gamma \\ 0 & \text{otherwise} \end{cases}$$

For a directed network, a -1 is entered for an oppositely directed branch.

2.3.3 Branch-capacity and Terminal Capacity Matrices

The other way of representing a communication net of n vertices is by using a symmetric matrix $C = [c_{i,j}]$ called the branch-capacity matrix where $c_{i,j} = c_{j,i}$ is the sum of the capacities of the branches connected between v_i and v_j. For an oriented network, the C-matrix is asymmetrical.

For any two vertices in $\overline{G}$, there exist a finite, maximum obtainable capacity of transferring information from one vertex to the other called the terminal capacity. The terminal capacity matrix $T = [t_{i,j}]$ represents the maximum flow that can exist between all the possible vertex pairs of $\overline{G}$. The T-matrix is also asymmetrical for a directed network.

2.3.4 Connectivity and Redundancy Matrices

The connectivity matrix is essentially a T-matrix with each branch of G or $\overline{G}$ assigned a maximum capacity of unity.

An n x n matrix $R = [r_{i,j}]$ is a redundancy matrix where the i-jth entry $r_{i,j}$ is the number of branch/vertex disjoint paths between v_i and v_j. It is symmetrical for a non-oriented net. A uniform R-matrix is one which has identical entries except the main diagonal.

The main diagonals of these matrices have infinity as entries however for computer calculation zero entries are used.

2.4 DISCONNECTING SETS OF A NETWORK

A connected network represented by G can be disconnected by deleting certain branches or vertices.

An s-t branch cutset is a set of branches whose deletion breaks all directed paths from v_s to v_t; likewise an s-t vertex cutset is a set of vertices whose deletion breaks all directed paths from v_s to v_t. A mixed s-t cutset is composed of branches and vertices.

Clearly if the capacity of each branch or vertex is set to unity, then the value of the cutset is numerically equal to the number of components of the cutset.

Let $\omega_{s,t}$ be the number of elements in the smallest s-t vertex cutset;

$\tau_{s,t}$ be the number of elements in the smallest s-t branch cutset;

$\sigma_{s,t}$ be the number of elements in the smallest s-t mixed cutset; then the disconnecting sets of the connected graph are as follows:

$$\omega = \underset{\substack{s,t \in V \\ s \neq t}}{\text{Min}} \left[\omega_{s,t} \right] \qquad (2.4.1.1)$$

$$\tau = \underset{\substack{s,t \in V \\ s \neq t}}{\text{Min}} \left[\tau_{s,t} \right] \qquad (2.4.1.2)$$

$$\sigma = \underset{\substack{s,t \in V \\ s \neq t}}{\text{Min}} \left[\sigma_{s,t} \right] \qquad (2.4.1.3)$$

The smallest disconnecting set of a connected graph denoted by ν is defined as

$$\nu = \text{Min} \left[\omega , \tau , \sigma \right] \qquad (2.4.1.4)$$

However it was shown in [3] that

$$\nu = \omega = \underset{\substack{s,t \in V \\ s \neq t}}{\text{Min}} \left[\omega_{s,t} \right]. \qquad (2.4.1.5)$$

III. ANALYSIS OF DETERMINISTIC SURVIVABLE NETWORKS

3.1 SURVIVABILITY CRITERIA

A significant criterion in network analysis is its survivability or vulnerability. In a communication net, this parameter refers to the degree in which the system remain functional after an enemy attack or natural disaster.

No general survivable criteria has yet been formulated however several measures of survivability has been investigated namely:

1. The number of links and/or stations that must be destroyed before communication is disrupted [4],

2. the minimum number of links that must be deleted in order to isolate a group of stations from communication with the remainder of the net [4],

3. the minimum number of isolated subnets that would result from an attack or destruction. This is termed the "independence number" of the network [1],

4. fraction of stations which survive an attack [2], [6],

5. fraction of stations that can be reached from a given point by a path of no more than r links after an attack [8].

The first three criteria apply to deterministic network while the last two criteria are associated with probabilistic net.

Analysis of large networks based on the first criterion is treated in this thesis. This index of survivability suggests the application of the concept of disconnecting sets. More specifically, the parameter ω is a measure of vulnerability to station destruction and τ is a measure to link destruction.

3.2 ANALYSIS WITH RESPECT TO BRANCH DISCONNECTION

3.2.1 s-t Branch Connectivity

The problem discussed here is the finding of the maximum number of branch disjoint s-t paths between any given pair of vertices v_s and v_t in G. Setting the branch capacities to unity and applying the max-flow min-cut theorem, the maximal flow $f_{s,t}$ from v_s to v_t is obtained which is numerically equal to the smallest branch cutset $\mathcal{T}_{s,t}$. This value further represents the maximum branch disjoint paths between the given vertex-pair.

The Ford-Fulkerson algorithm solves a general max-flow problem between two vertices, the source and sink. It is stated here in simplified form to adapt it to the special case being investigated.

A few terms are defined below before stating the algorithm:

1. A vertex is labeled by an ordered pair $(i,\pm)$. For a particular vertex v_j, the index i denotes the preceeding labeled vertex v_i. The second term in the label is plus if for a forward branch, $f(i,j) = 0$ and a minus sign if for a backward branch, $f(j,i) > 0$. Otherwise a vertex is unlabeled.

2. A particular vertex v_x is scanned if its adjacent vertices v_y are labeled from it. Otherwise v_x is unscanned.

3. A vertex is considered to be in one of the three states, namely:

   a. unlabeled and unscanned

   b. labeled and unscanned

   c. labeled and scanned

The following steps determines the s-t branch connectivity of a directed or mixed network:

1. Represent the network by a directed linear graph. Number the vertices consecutively from 1 to n.

2. For each branch $(i,j) \in \Gamma$, assign $c(i,j) = 1$ and $f(i,j) = 0$.

3. Choose the source and sink vertices v_s and v_t.

4. Apply Algorithm 1

Algorithm 1 (Modified Ford and Fulkerson Algorithm)

Labeling Routine:

a. Erase all vertex labels

b. Label v_s by $(0,+)$. Source is now labeled and unscanned, all other vertices are unlabeled and unscanned.

c. Select any labeled unscanned vertex v_i (including v_s). Assign labels to all unlabeled vertices v_j adjacent to v_i such that

1) for forward branch (i,j) with flow $f(i,j) = 0$, label v_j by $(i,+)$. Vertex v_j is now labeled and unscanned.

2) for backward branch (j,i) with flow $f(j,i) > 0$, label v_j by $(i,-)$. Vertex v_j is now labeled and unscanned.

With all adjacent vertices to v_i labeled and unscanned, v_i is now scanned.

d. Repeat Step c until either

1) v_t is labeled or

2) no more labels can be assigned and v_t is unlabeled.

In Step d(1), an augmentation path is found and an increase of flow is possible. Proceed to Augmentation Routine.

In Step d(2), the algorithm terminates. Flow is now maximized. The disconnecting cutset are the branches joining the labeled and unlabeled vertices. The value of the cut is the branch connectivity.

Augmentation Routine:

a. Let $z = t$ and go to Step b.

b. If label on v_z is $(q,+)$, set $f(q,z) = 1$. If label on v_z is $(q,-)$, set $f(z,q) = 0$.

c. If q = s return to Step a of the Labeling Routine. Otherwise set z = q and go to Step b of the Augmentation Routine.

Example: A mixed network

Fig. 3.1(a) is the linear graph representation of the network. In (b), the non-oriented branch (3,6) is split into two oppositely directed branches (3,6) and (6,3). The first labeling obtained the augmentation path s,(s,3),3,(3,7),7,(7,t). In (c) the second labeling found the augmentation path s,(s,4),4,(4,3),3,(3,6),6,(6,t). In (d) the sink vertex v_t cannot be labeled since (6,8) and (7,8) are already saturated and (5,8) is a backward branch with zero flow. The algorithm terminates. The cut is $\{(6,8),(7,8)\}$ and s-t connectivity is two.

Flow chart for this analysis is in Appendix A-1 and the computer program MAXCON-1A is in Appendix B-2. To minimize computer storage, a vertex label is further modified by using a single signed integer ($\pm i$) instead of the ordered pair ($i,\pm$). When i is negative the magnitude is obtained by just multiplying by minus 1.

In the program, vertex scanning is accomplished by using two storage vectors. The first vector is a list of the labeled vertices being scanned while the second provides storage for the vertices being labeled. Once the scanning process of all the vertices listed in the first vector is completed, the contents of the second is transferred to the first which in turn becomes the list of vertices to be scanned.

Algorithm 1 is applied to the analysis of undirected network with slight modification. For a non-oriented net, each branch flow is considered bidirectional with a maximum capacity of unity. However, in no case can a positive back-flow occur since each augmentation path has as its elements all forward directed branches with respect to v_s and v_t.

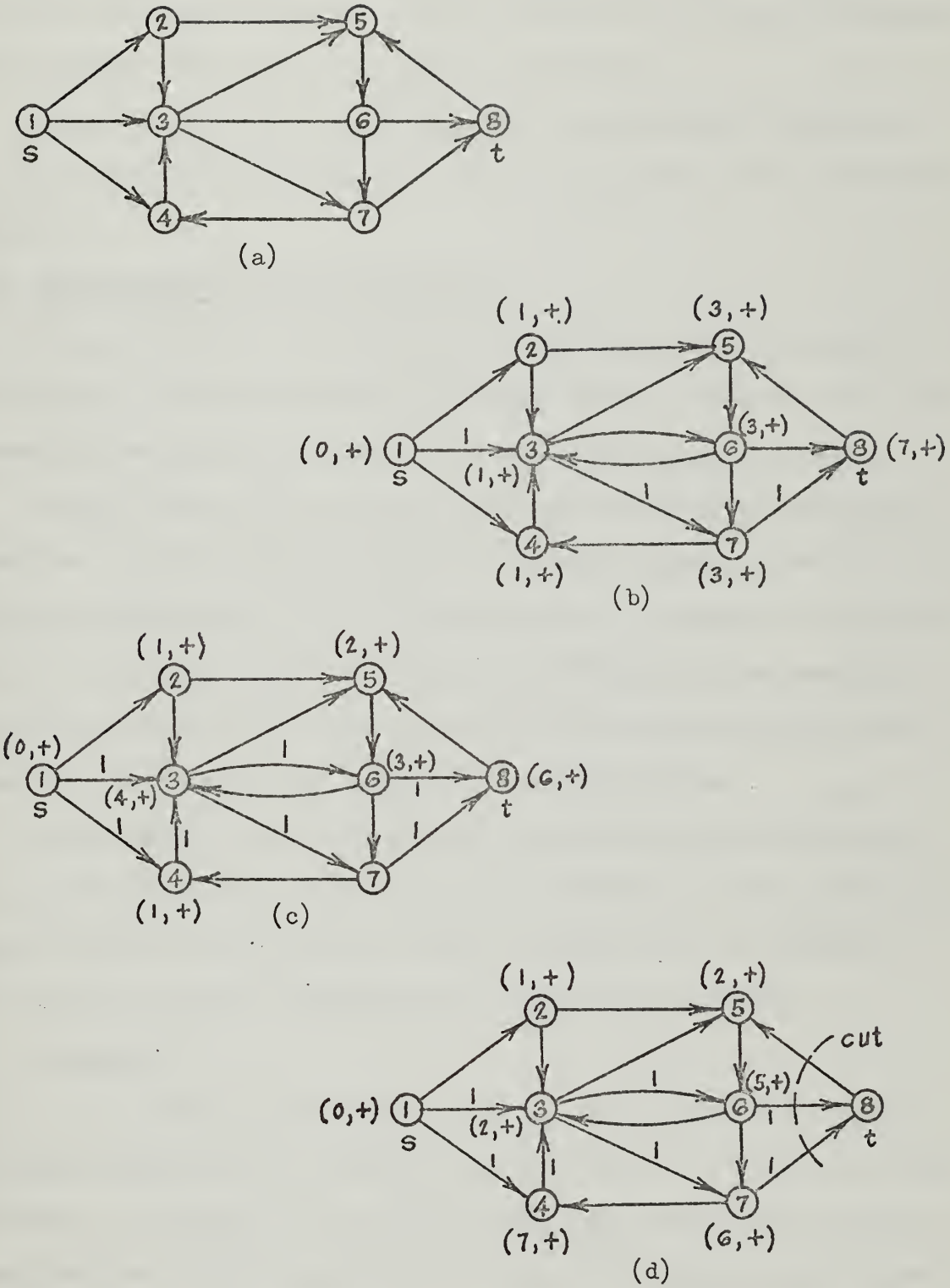


Fig. 3.1

Consequently there cannot be any negatively labeled vertex. Therefore the tests for a backward branch and a negative-labeled vertex, and the adjustment of backward flow are not necessary in this case.

These steps are deleted in Algorithm 1 and MAXCON-2A (Appendix B-4) is developed for determining s-t branch connectivity of an undirected network.

3.2.2 Multiterminal Branch Connectivity

For an n-station network, its graph would have $n(n-1)/2$ possible vertex pairs. To apply Algorithm 1 $n(n-1)/2$ times in order to obtain the connectivity between all the possible vertex-pairs would be lengthy.

Gomory and Hu [13] have shown that for an undirected network with n vertices, any flow in the network is numerically equal to some flow in a maximal spanning tree. Since there are only n-1 branches in a spanning tree, then there can only be n-1 numerically different flows possible. Therefore all the $n(n-1)/2$ possible maximal flows that can exist in the network can be deduced after doing n-1 flow maximizations.

Applying the above principle and the max-flow min-cut theorem in the analysis of non-oriented nets, the multi-terminal network connectivity problem can be solved by only n-1 separate applications of Algorithm 1. The analysis procedure is embodied in the following algorithm.

Algorithm 2

1. Choose two vertices v_s and v_t in the undirected graph $\bar{G}$. With branch capacities set to unity, find an initial s-t branch cut using Algorithm 1. Represent this cut by a generalized vertex tree with condensed vertices X and $\bar{X}$ connected by a branch with capacity equal to the computed cut value $\tau_{s,t}$.

2. Draw a condensed network graph $\bar{G}_1$ where $\bar{X}$ is represented by a single vertex. Select two vertices v_k and v_l in X and find the

smallest k-1 cut by Algorithm 1. Form the generalized vertex tree by dividing X into X_1 and $\bar{X}_1$ with τ_{k-1} as the capacity of the branch connecting them. In this tree, X is adjacent to X_1 if it is in the same side of the cut as X_1, or X is adjacent to $\bar{X}_1$ if it is in the same side of the cut as $\bar{X}_1$.

3. The above procedure is repeated. With each step, a tree of generalized vertices connected by branches with corresponding cut values is obtained. To proceed with the computation,

a. Select a generalized vertex X_i

b. Condense each component in $\bar{G}$ except X_i into a single vertex forming the condensed network graph $\bar{G}_i$.

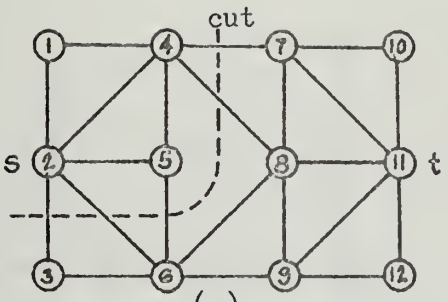
c. Choose two original vertices v_s and v_t in X_i as source and sink and apply Algorithm 1 to $\bar{G}_i$ to get $\tau_{s,t}$.

The cut obtained in this process splits X_i into two parts X_{i_1} and X_{i_2}. This is represented in the generalized vertex tree by replacing X_i by the condensed vertices X_{i_1} and X_{i_2} connected by a branch bearing the cut value. All other branches and vertices in the tree are unchanged except those which were formerly connected to X_i. Apply the adjacency rule as in Step 2.

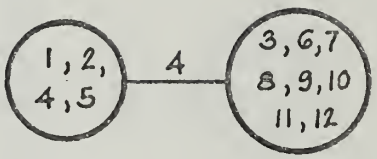
4. Repeat the process until these generalized vertices consist of exactly one vertex each. This results in a spanning tree. Thus for any chosen v_s and v_t, the s-t connectivity is the smallest branch weight in the unique s-t path in the tree.

An example for the application of the above procedure is given in Fig. 3.2. The generalized vertex tree and the condensed network $\bar{G}_i$ resulting in each flow calculation are indicated. For the 12-vertex network, the spanning tree is formed after 11 max-flow computations.

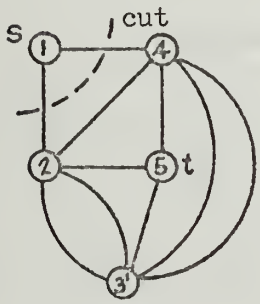
Fig. 3.2



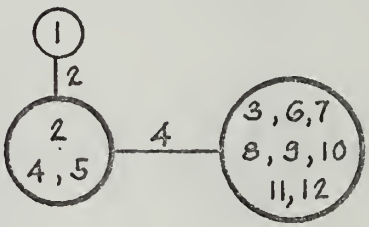
(a)



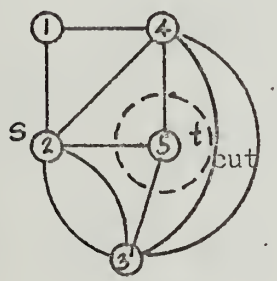
(b)



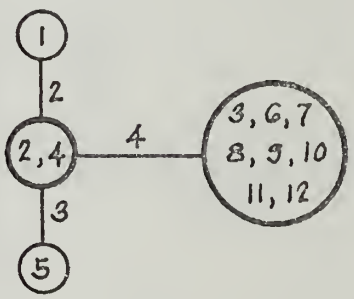
(c)



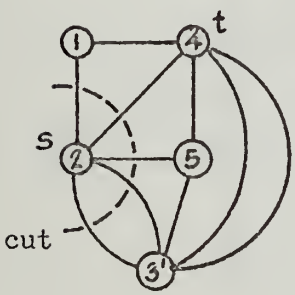
(d)



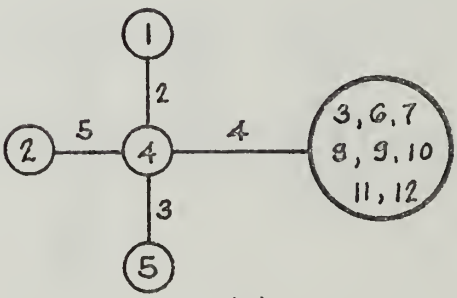
(e)



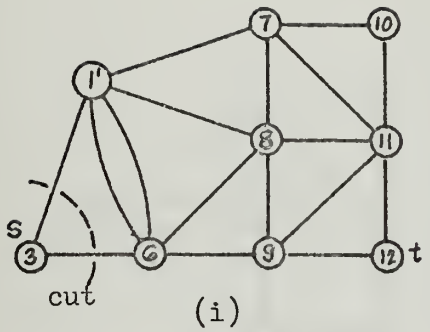
(f)



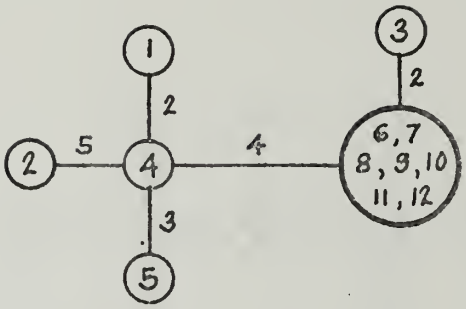
(g)



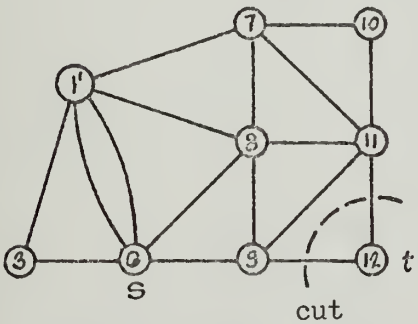
(h)


7
10
1'
8
11
s
3
6
9
12
t
cut

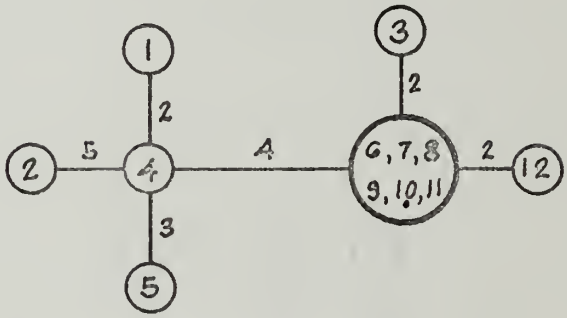
(i)


1
3
2
2
5
4
4
6,7
8,9,10
11,12
3
5

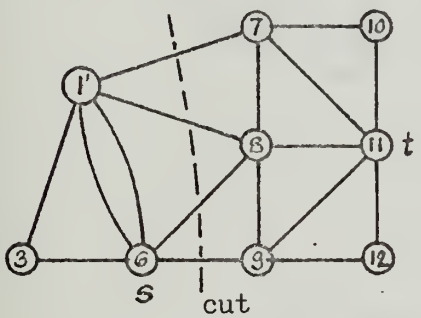
(j)


7
10
1'
8
11
3
6
9
12
t
s
cut

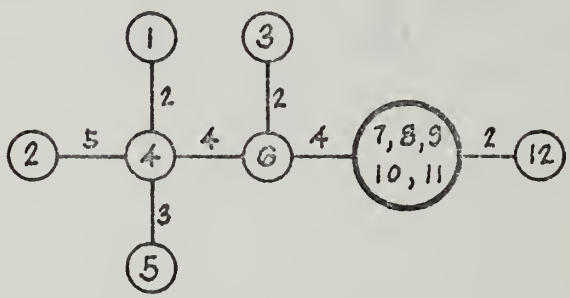
(k)


1
3
2
2
5
4
4
6,7,8
9,10,11
2
12
3
5

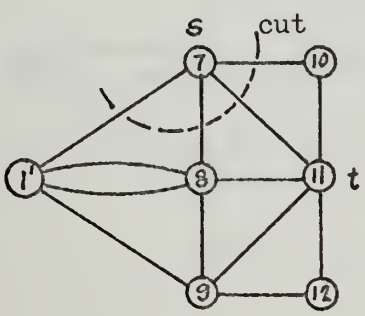
(l)


7
10
1'
8
11
t
3
6
9
12
s
cut

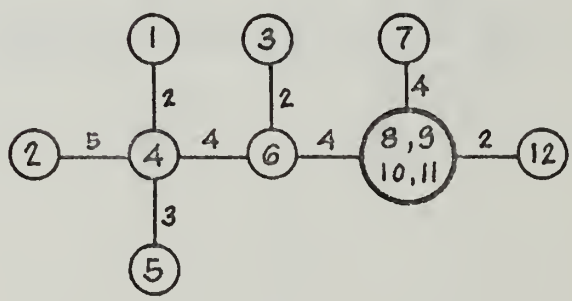
(m)


1
3
2
2
5
4
4
6
4
7,8,9
10,11
2
12
3
5

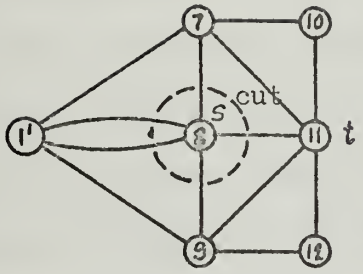
(n)


s
cut
7
10
1'
8
11
t
9
12

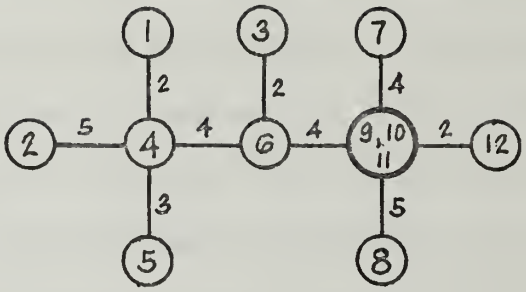
(o)


1
3
7
2
2
4
5
4
4
6
4
8,9
10,11
2
12
3
5

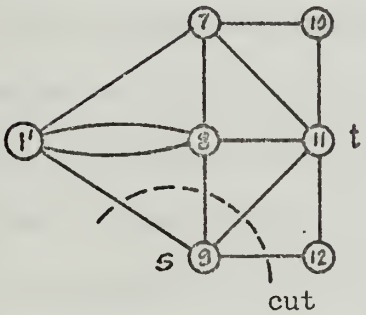
(p)


7
10
1'
8
s
cut
11
t
9
12

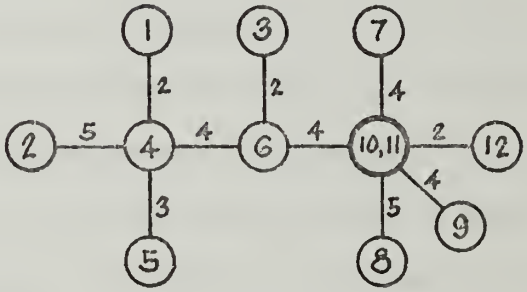
(q)


1
3
7
2
2
4
2
5
4
6
4
9,10
11
2
12
3
5
5
8

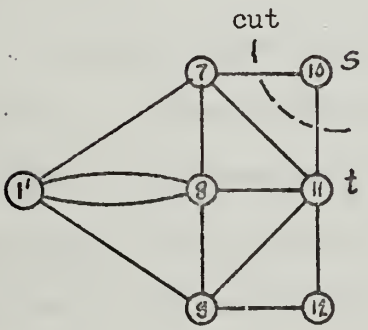
(r)


7
10
1'
8
11
t
s
9
12
cut

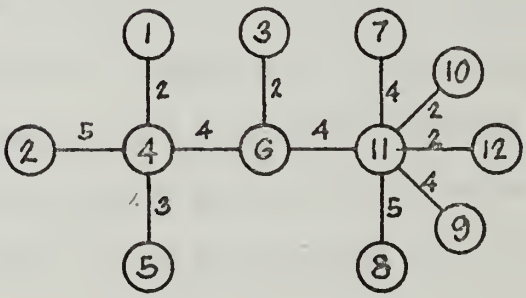
(s)


1
3
7
2
2
4
2
5
4
6
4
10,11
2
12
4
3
5
9
5
8

(t)


cut
7
10
s
1'
8
11
t
9
12

(u)


1
3
7
10
2
2
4
2
5
4
6
4
11
2
12
4
3
5
9
5
8

(v)

A flowchart for this analysis procedure is given in Appendix A-2. This is further encoded into a computer program entitled MAXCON-2 appearing in Appendix B-3. In this program the printout is not a spanning tree but rather a symmetrical branch-connectivity matrix which is essentially a terminal capacity matrix. Its entries are derived from the cut values appearing as branch weights of the trees generated after every max-flow calculation where a cut isolates a single vertex from the rest of the condensed network graph $\overline{G}_i$. The connectivity between this single generalized vertex and all the rest of the vertices in $\overline{G}$ are the updating entries.

Other information relevant to MAXCON-2 are stated in the program documentation.

The concept of spanning tree cannot be applied to the analysis of a directed or mixed network as generally these networks are not symmetrical. To obtain a multiterminal branch-connectivity matrix for an oriented net, it is necessary to do $n(n-1)/2$ flow maximizations by Algorithm 1. Up to present, there is no available method that could reduce this number of calculations however Frisch has developed the Flow Variation Algorithm [12] which effectively reduces the number of steps required to maximize $f_{s,t}$.

The ideas underlying this new algorithm are as follows: let v_s, $v_{s'}$, and v_t be three arbitrary vertices in a graph G. Rather than apply Algorithm 1 to independently maximize $f_{s',t}$ and $f_{s,t}$, first maximize $f_{s',t}$ by Algorithm 1 then maximize $f_{s,t}$ in the usual manner until a vertex v_g is labeled for which $f(g,V) > 0$. Then use the flow pattern $D(s',t)$ to find $F(s,t)$.

Applying this basic idea to solve the multiterminal connectivity problem which is a special case of a general flow problem, all that is necessary is to slightly modify the algorithm to fit into the problem.

The result is a simplified version of the original algorithm. The following notations are used:

$\pi_{s,t}$ - flow path for vertex-pair v_s and v_t

$\pi_{s',t}$ - flow path for vertex-pair $v_{s'}$ and v_t

$F(s,t)$ - flow pattern for vertex-pair v_s and v_t

$D(s',t)$- flow pattern for vertex-pair $v_{s'}$ and v_t

$f(i,j)$ - a branch flow in path $\pi_{s,t}$

$d(i,j)$ - a branch flow in path $\pi_{s',t}$

$\pi^A_{s,g}$ - section of path $\pi_{s,t}$ from v_s to v_g called "Augmentation Path"

$\pi^T_{s',g}$ - section of path $\pi_{s',t}$ from $v_{s'}$ to v_g called "Truncation Path"

$\pi^H_{g,t}$ - section of path $\pi_{s',t}$ from v_g to v_t called "Homing Path"

$L^H_{m,m}$ - a loop detected by the Homing Routine; m is the first vertex in the loop

$L^T_{p,p}$ - a loop detected by the Truncation Routine; p is the first vertex in the loop

The subscript c whenever it appears with the symbols indicates the c^{th} stage of max-flow calculation.

The following routines are employed in the algorithm:

Routine A - Detecting an Augmentation Path

1. Label v_s by (s,+). v_s is now labeled and unscanned. All other vertices are unlabeled and unscanned.

2. Select any labeled and unscanned vertex v_i. Assign labels to all unlabeled vertices adjacent to v_i such that

a. for forward branch $(i,j) \in \Gamma$ with flow $f(i,j) = 0$, label v_j by $(i,+)$. v_j is now labeled and unscanned.

b. for backward branch $(j,i) \in \Gamma$ with flow $f(j,i) > 0$, label v_j by $(i,-)$. v_j is now labeled and unscanned.

With all adjacent vertices to v_i labeled and unscanned, v_i is now scanned.

3. Repeat Step 2 until either:

a. v_t is labeled or

b. a vertex v_g is labeled with $d_c(g,V) > 0$

c. no more vertices can be labeled and v_t is unlabeled.

Observe that except for Step 3(b) this routine is the Ford and Fulkerson Labeling Routine.

Routine T - Detecting a Truncation Path

1. Let $z = g$

2. Find one vertex v_i such that $d_c(i,z) > 0$ and label v_i by (z)

3. Let $z = i$

a. If v_z has assumed a value i before, terminate.

b. If $i = s'$, terminate.

c. Otherwise return to Step 2.

If the routine terminates in Step 3(a), a loop $L^T_{p,p}$ has been detected rather than a path.

Routine H - Detecting a Homing Path

1. Let $q = g$

2. Find one vertex v_j such that $d(q,j) > 0$ and label v_j by (q)

3. Let $q = j$

a. If q has assumed the value j before, terminate.

b. If $j = t$, terminate.

c. Otherwise return to Step 2.

If the routine terminates in Step 3(a), a loop $L^H_{m,m}$ has been detected instead of a path.

The main algorithm for obtaining the multiterminal branch-connectivity of an oriented network is now stated in terms of Routines A, T, and H.

Algorithm 3

1. Choose two vertices v_s and v_t. Using Routine A, find an initial s-t cut.

2. Using Routine A find an s-g augmentation path $\pi^A_{s,g}$ or an s-t augmentation path $\pi^A_{s,t}$. In the former case go to Step 3. In the latter case go to Step 8. If no such path can be found, terminate.

3. Using Routine H find a Homing Path $\pi^H_{g,t}$ or a Homing Loop $L^H_{m,m}$. In the former case go to Step 4; in the latter case go to Step 5(a).

4. Using Routine T find a Truncation Path $\pi^T_{s',t}$ or a Truncation Loop $L^T_{p,p}$. In the former case go to Step 6; in the latter case go to Step 5(b).

5. a. Set to zero all $d_c(i,j)$ for all branches (i,j) in the loop $L^H_{m,m}$. Erase all vertex labels and go to 2.

b. Set to zero all $d_c(i,j)$ for all branches (i,j) in the loop $L^T_{p,p}$. Erase all vertex labels and go to 2.

6. Trace and erase g-t path. For every branch (i,j) in $\pi^H_{g,t}$, set

$$f_{c+1}(i,j) = 1$$

$$d_{c+1}(i,j) = 0$$

7. Truncate the s'-g path. For every branch (i,j) in $\pi^T_{s',g}$, set

$$d_{c+1}(i,j) = 0.$$

8. Augment the s-g path. For every branch (i,j) in $\pi^A_{s,g}$, set

$$f_{c+1}(i,j) = 1.$$

Erase vertex labels and return to Step 2.

If a "loop of flow" is detected, the flow in this loop must be set to zero as in Step 5 as otherwise a nonfeasible flow will be achieved.

As an example, consider the graph in Fig. 3.3. Let the indicated flow pattern D(s',t) be the result of initial maximization of $f_{s',t}$. Designate v_2 as the source for the second stage of maximization. Step 2 of the algorithm labels v_3 and detects it as a vertex v_g. Step 3 finds the sequence 3,(3,6),6,(6,8),8 as the Homing Path $\pi^H_{g,t}$. The path s',(1,3),3 is the Truncation Path $\pi^T_{s',g}$, and s,(2,3),3 the Augmentation Path $\pi^A_{s,g}$. Step 6 sets to zero the branch flows $d_c(i,j)$ from v_3 to v_t and replaces each branch flow in the path by $f_{c+1}(i,j) = 1$. Step 7 truncates s'-3 path by setting $d_c(1,3)$ to zero, and Step 8 augments s-3 path by setting $f_{c+1}(2,3) = 1$. One s-t path is now processed and the algorithm will search for another one until no more augmentation paths can be found. By then, $f_{s,t}$ will be maximal.

Flowchart of this algorithm is in Appendix A-3 and its computer implementation MAXCON-1 is presented in Appendix B-1.

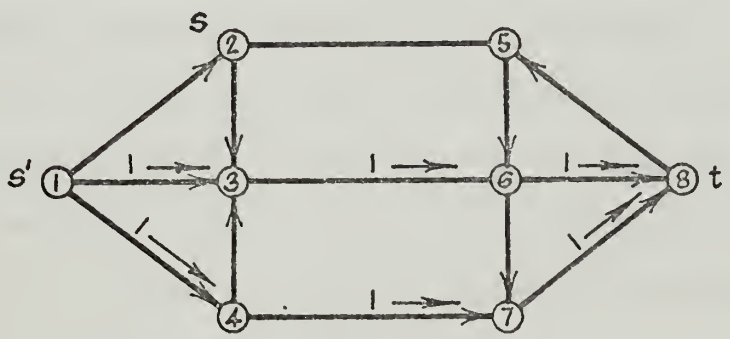
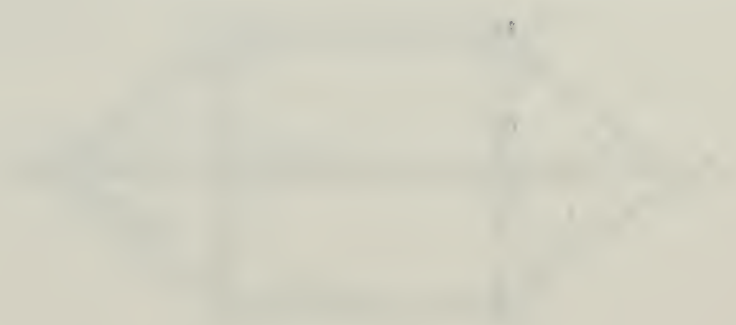


Fig. 3.3



3.3 ANALYSIS WITH RESPECT TO VERTEX DISCONNECTION

The criterion of survivability attached to this problem is the number of stations that must be deleted from the network in order to disrupt communication. This is the smallest disconnecting set ω, of a network graph. Given any pair of vertices v_s and v_t, assign unit capacities to all vertices except v_s and v_t (which have infinite capacities), then the smallest s-t vertex cutset $\omega_{s,t}$ is numerically equal to the maximum number of vertex disjoint s-t paths. The value of $\omega_{s,t}$ is the network vertex-connectivity with respect to the given source and sink.

3.3.1 s-t Vertex Connectivity

To solve the problem of finding $\omega_{s,t}$, each vertex in G is split into two vertices connected by a branch of unit capacity forming a new graph G'. Algorithm 1 is then applied to G'. The result is a branch cutset composed only of the unit capacity branches. The regular network branches are considered having infinite capacities. This approach is however very inefficient since the number of vertices are almost doubled.

An algorithm known as Vertex-Pair Connectivity derived from Algorithm 1 was developed by Frisch [11] which applies directly to graphs of directed, undirected, or mixed networks without splitting the vertices. The vertex-splitting process is absorbed in the modification of the labeling routine of Algorithm 1. The algorithm is stated here in a different form. Several terms and rules must first be defined:

1. A vertex is labeled with an ordered triple $(i,\pm,M)$. The first and second entries are defined as in the original Labeling Routine. The third entry is a zero or one as will be explained in the algorithm.

2. A vertex is strongly labeled if the second entry is a minus, and weakly labeled if it is a plus.

3. A circle around the second entry denotes the vertex is scanned.

4. If a vertex is weakly labeled and unscanned, and then receives a strong label, the weak label is erased. However if the label is scanned, it should not be erased.

Algorithm 4

Labeling Routine

1. Erase vertex labels. Label v_s by $(s,+,0)$. v_s is now labeled and unscanned. All other vertices are unlabeled and unscanned.

2. Select any labeled unscanned vertex v_i. If none, terminate.

a. Suppose v_i is weakly labeled:

1) If $f(i,j) = 0$, $(i,j) \in \Gamma$ and $j \neq s$, weakly label all vertex v_j by $(i,+,0)$. v_j is now weakly labeled and unscanned.

2) If $f(j,i) = 1$, $(j,i) \in \Gamma$, $j \neq s$, and v_j not strongly labeled, strongly label v_j by $(i,-,0)$. v_j is now strongly labeled and unscanned.

Scan v_i by encircling the + in the v_i label.

b. Suppose v_i is strongly labeled

1) If $f(i,j) = 0$, $(i,j) \in \Gamma$, and $j \neq s$, weakly label all unlabeled vertices v_j by $(i,+,M)$. v_j is now weakly labeled and unscanned.

2) If $f(j,i) = 1$, $(j,i) \in \Gamma$, $j \neq s$, and v_j not strongly labeled, strongly label v_j by $(i,-,M)$. If v_j is weakly labeled, erase the weak label, v_j is now strongly labeled and unscanned.

In (b), if v_i is both weakly and strongly labeled, set $M = 1$, otherwise $M = 0$. Scan v_i by encircling the minus sign in the v_i label.

3. If v_t is labeled, proceed to Augmentation Routine. Otherwise return to Step 2.

Augmentation Routine:

1. Let $z = t$

2. a. If v_z is weakly labeled by $(q,+,M)$, set $f(q,z) = 1$.

   b. If v_z is strongly labeled by $(q,-,M)$, set $f(z,q) = 0$.

3. If $q = s$ return to Step 1 of Labeling Routine. Otherwise proceed to Step 4.

4. Let $z = q$. If $M = 1$ return to Step 2(b). Otherwise return to Step 2(a).

To identify the s-t vertex cutset, let U be the set of all unlabeled vertices, and L be the set of all labeled vertices. Further let L_u be the set of labeled vertices connected to an unlabeled vertex by a branch directed from a labeled vertex to an unlabeled one, then the s-t vertex cut X_m is given by:

$$X_m = L_u \quad \text{if } s \notin L_u \tag{3.3.1.1}$$

$$X_m = (L_u - v_s) \cup \left\{v_h : (s,h) \in (L,U)\right\} \quad \text{if } s \in L_u \tag{3.3.1.2}$$

Since the source vertex cannot be a component of the vertex cut, Eq. (3.3.1.2) shows that if v_s is connected to unlabeled vertices v_h, then the vertices v_h are some component of the cut.

The graph in Fig. 3.4 illustrates the application of the algorithm for a directed network. Assume that the path indicated by solid arrows is a result of initial labeling and augmentation. Assume further that the second application of labeling routine is just completed and v_t is labeled as indicated by the vertex labels. When the augmentation routine is applied, it will backtrack the path in the usual manner, however, on reaching v_4 it has to select the strong label since v_6 has $M = 1$. If the weak label were selected, a non-feasible flow would result. This explains the purpose of the third entry in the label. The only case where two

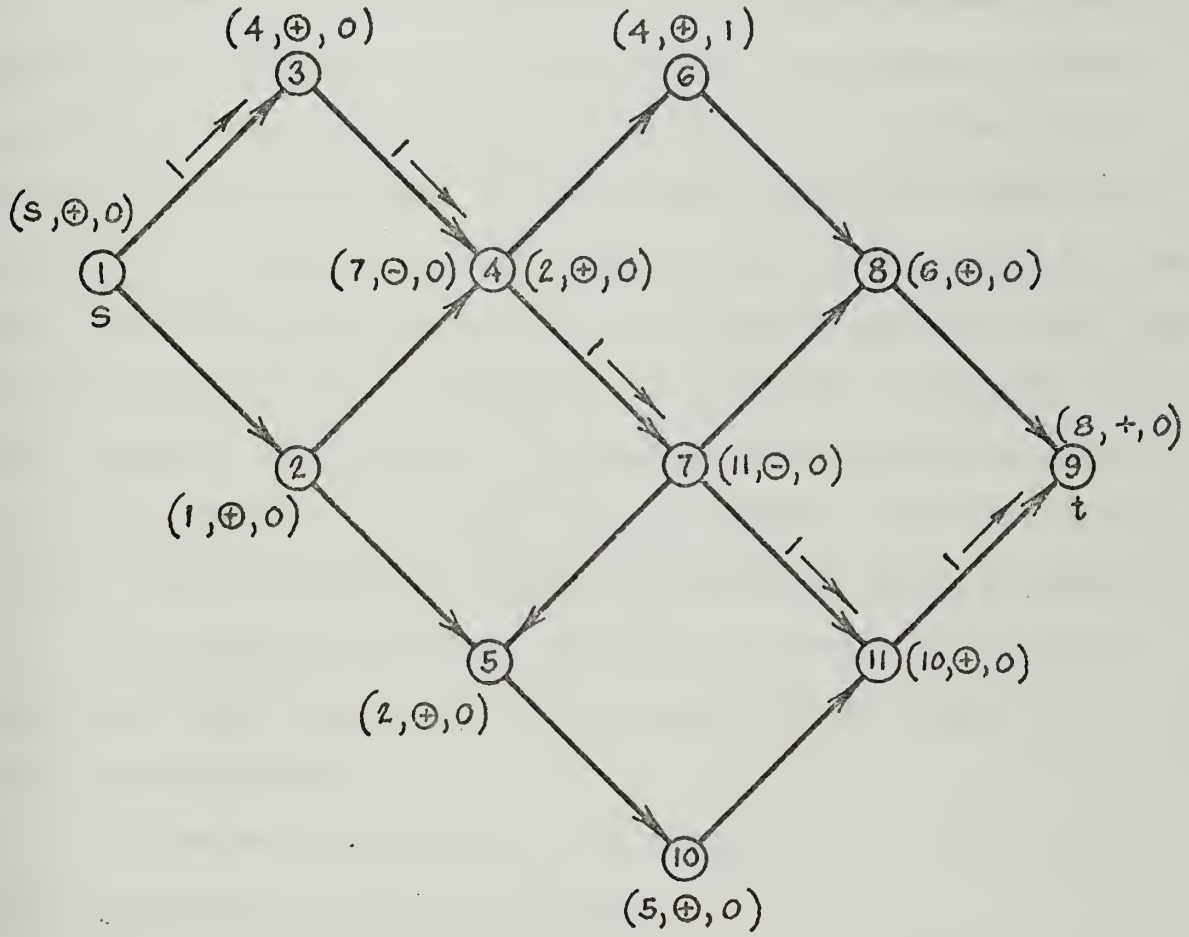


Fig. 3.4

scanned labels can occur on a vertex is the situation in v_4 where a flow of unit magnitude already exists through the vertex and this vertex receives labels in the succeeding iteration of the algorithm. If the weak label in v_4 was unscanned when it received a strong label then the weak label should have been erased and v_6 would have been labeled (4,+,0).

Program MAXCON-3A in Appendix B-6 is a computer implementation of Algorithm 4. A different way of circumventing the situation in vertex v_4 of Fig. 3.4 was applied. For a vertex like v_4 where a flow already exists, if in the succeeding labeling process v_4 receives a weak label, it will not be scanned, but if it receives a strong label it will be scanned. The latter would indicate that the existing outward flow from v_4 can in effect be diverted to another path. In the former, it would be wasteful to continue seeking another path passing through v_4 when the vertex is already saturated. In no case then can a vertex acquire two scanned labels. The third entry in the label is therefore no longer necessary.

The same form of label as in MAXCON-1A is used in this program. All steps in the algorithm that are associated with the third entry in the label are deleted.

A flowchart of Algorithm 4 is in Appendix A-4.

3.3.2 Multiterminal Vertex Connectivity

The Flow Variation Algorithm of Frisch as presented in Sec. 3.2 maximizes flows in a finitely branch-weighted network, and it was readily adapted to solving the multiterminal branch-connectivity. In this section the problem is different in the sense that the vertices control the information flow.

In order to utilize the principle of the Flow Variation Algorithm here, the Routine A of the algorithm must be altered. Recall that Routine A is the Ford-Fulkerson Labeling Routine. If this routine is replaced by

the Labeling Routine of Algorithm 4 which is Frisch's Vertex-Pair Connectivity Algorithm, it would enable the Flow Variation Algorithm to handle the multiterminal vertex-connectivity problem. Consequently the method in which the s-g and s-t paths are augmented must also conform with the Augmentation Routine of Algorithm 4.

With these changes, the algorithm flowchart is the same as in Appendix A-3, however there is a significant change in the computer program MAXCON-3 as presented in Appendix B-5 compared to MAXCON-1.

The solution given earlier for obtaining the multiterminal vertex-connectivity of a directed network applies equally well for undirected networks. Sometimes one finds it impractical to compute all the entries in the entire connectivity matrix especially when investigating a very large network. It is often sufficient to just verify whether a network satisfies a certain minimum degree of connectivity.

With this analysis criterion it is only necessary to verify that the network is at least R-connected. The connectivity being referred to here is the vertex-connectivity which we know is the smallest.

The following theorems of Kleitman [14] greatly reduce the number of verifications necessary rather than doing $n(n-1)/2$ separate tests for the existence of R vertex-disjoint paths.

Theorem:

To verify that all $n(n-1)/2$ vertex pairs in G can be connected by at least R vertex-disjoint paths, it is only necessary to verify that some set of R vertices are each connected by at least R vertex-disjoint paths with every vertex in the graph.

Two other theorems resulting from the above are stated here in an algorithm form and are in themselves the analysis procedure.

Algorithm 5

1. Choose any vertex, say v_1 as source.

Let i = 1.

a. Select a vertex v_j in G_i as sink. Remove from G_i all other vertices $\{v_{i_k}\}$, k = 1,...,l previously designated as sink that are adjacent to v_j. If $l \geq R$ go to Step 1(b).

1) For v_j connected to v_i

Excluding the direct path i-j, use Algorithm 4 to verify existence of q vertex-disjoint paths from v_i to v_j. Set s = q+1. If s < R go to Step 4. Otherwise go to Step 1(b).

b. Are all n-i vertices processed? If not go to Step 1(a), otherwise go to Step 2.

2. Decrease R by i. If R = 0, terminate; otherwise go to Step 3.

3. Remove v_i from G_i. Choose vertex v_{i+1} as source. Increase i by one. Return to Step 1(a).

4. Record s-connectivity between v_i and v_j. Terminate. Network is not R-connected.

The first theorem may be restated in a more specific sense:

If there are at least R vertex-disjoint paths between v_1 and v_i and v_1 and v_j in G, then to verify the existence of R such paths between v_i and v_j in G, we need only look for R - 1 such paths in $G - \{v_1\}$.

For the proof of these theorems and illustrative example, the reader is referred to the original paper by Kleitman.

A flowchart for computer implementation is given in Appendix A-5. Program MAXCON-4 in Appendix B-7 is encoded to analyze an undirected network for a uniform R-connectivity where the value of R is initially specified.

IV. DESIGN OF LOW-COST SURVIVABLE NETWORK

4.1 INTRODUCTION

There are now a number of synthesis procedures for survivable networks appearing in literatures elsewhere. These are summarized by Frank and Frisch in [10] and are categorized according to link or station damage. The synthesis approach are all aimed towards maximization of redundancy between stations and satisfying network redundancy requirement with the assumption that cost is either uniform for each link/station or not considered at all.

In the design of practical flow networks, the cost of construction increases with increase in redundancy therefore the design approach must take into account non-uniform cost for the construction of links or stations. The goal is a minimum cost network that satisfies a prespecified redundancy requirement. So far there is no exact analytical method that could be applied to solve the problem. A procedure to search for a low-cost network with respect to vertex disconnection is presented by Steiglitz, etal [18] using an heuristic approach. The algorithms they developed were based on the techniques for an approximate solution to the traveling salesman problem. The salesman's tour is a Hamiltonian circuit which is a special case of the flow network having a redundancy of two.

4.2 DESIGN WITH BRANCH DISCONNECTION CRITERION

In some cases branch destruction is more likely than vertex destruction and it is just as necessary to obtain a design method using the former criterion as the latter. A procedure is presented in this chapter for design of a low cost network (with high probability of being optimal) with at least $r_{i,j}$ branch disjoint paths.

The same procedure used by Steiglitz, etal is employed consisting of two parts. The first part called the Starting Routine generates a feasible initial network (one which satisfies redundancy requirement) and the second part is the Optimizing Routine which makes an iterative improvement on the network cost by branch exchanges while maintaining feasibility. Portions of the procedure that differ from that of Steiglitz are the feasibility testing of the network and the optimizing routine. The method of optimization that will be presented is an adaptation of the recently published heuristic algorithm for the traveling salesman problem by Lin and Kernighan [16].

The heuristic design algorithm is stated as follows:

Algorithm 6

1. Generate a feasible initial network G that satisfies the specified redundancy R using the Starting Routine.

2. Attempt to find an improved feasible solution G' by some local transformation using the Optimizing Routine.

3. If a similar network with a lower cost is found, then replace G by G' and repeat from Step 2.

4. If no local improvement can be found, G is a locally optimal solution. Repeat from Step 1 until computation time runs out or the solution is satisfactory.

4.2.1 The Starting Routine

This routine was developed by Steiglitz, etal in [18].

1. Plot vertices representing stations in a Euclidian 2-space.

2. Number the vertices uniformly at random.

3. Assume cost to be proportional to distance and obtain the cost matrix from the Euclidian distances between vertices truncated to an integer.

4. Assign redundancy to each vertex.

5. Add branches to the network one at a time. Reduce by unity the assigned vertex requirement every time a link is connected to that particular vertex. A branch is added in this manner:

a. Connect the vertex with highest updated requirement to another vertex with the next highest updated requirement; for all vertices having equal requirement, choose one that results in lower cost.

b. In cases of ties in requirement and cost, these are resolved by choosing the vertex highest on the list.

c. No parallel branches are allowed.

6. When the requirement in all the vertices are reduced to zero, test feasibility of the initial solution using Algorithm 2.

7. If solution is feasible go to Optimizing Routine, otherwise repeat from Step 2.

This routine is made nondeterministic by numbering the vertices uniformly at random at each start. Randomized starting solution has been observed experimentally to produce a variety of starting networks as contrasted to constructive solutions which are deterministic where more than one solution may not be obtained.

The analysis tools presented in Chapter III come in handy for testing feasibility. For the initial network, the one that fits is Algorithm 2 which determines the overall branch connectivity of the network. A complete checkout need not be carried any further once infeasibility is detected.

4.2.2 Optimizing Routine

The optimizing procedure applied by Steiglitz is the X-change where two branches are interchanged at a time, and determined whether the transformation is favorable and preserves feasibility. Lin and Kernighan have

lately experimented on a more systematic and effective way of branch transformation in a network with 2-redundancy. Their results are applied here for a network with redundancy greater than two.

The basic ideas underlying the transformation are as follows: Consider a nonoptimal but feasible network G with cost c(G) and an optimal or near optimal network G' with the same number of vertices and with cost $c(G') < c(G)$. G is nonoptimal because it has k links $x_1,\ldots,x_k$ that are "out of place". To make G optimal they should be replaced by k links $y_1,\ldots,y_k$ of G'. We try to find k by identifying sequentially, element by element, two disjoint sets of links $X = \{x_1,\ldots,x_k\}$ and $Y = \{y_1,\ldots,y_k\}$ using a gain criterion. Let the costs of x_i and y_i be $|x_i|$ and $|y_i|$ respectively. Define $g_i = |x_i| - |y_i|$ as the gain from exchanging x_i with y_i. For some i, g_i may be negative however we want the final gain to be positive, i.e., $\sum_i^k g_i > 0$. We are looking for sequences of g_i's whose partial sum is always positive. When $\sum_i^k g_i \leq 0$, the search is stopped.

The k links in X are deleted or "broken" and replaced by k links in Y resulting in G' which has lower cost. The process is iterated until no further reduction in cost is attained.

We now state the routine more formally assuming that a starting solution has been generated:

1. Set P* = 0. (P* is the best improvement so far). Choose any vertex as t_1 and let x_1 be one of the edges of G incident to t_1. Let i = 1.

2. Designate the other endpoint of x_1 as t_2. From t_2 choose y_1 different from existing links in G such that $g_1>0$ and designate the other endpoint of y_1 as t_3. If no such y_1 exists, go to Step 5(d).

3. Let i = i + 1. Choose x_i (which currently joins t_{2i-1} to t_{2i}) and y_i as follows:

a. x_i is chosen so that it is the longest link in G incident to vertex t_{2i-1}

b. y_i is some link that can be drawn from vertex t_{2i} subject to (c), (d), and (e) below. If no y_i exists, go to Step 4.

c. To guarantee that x's and y's are disjoint, x_i cannot be a link previously joined (i.e., a y_j, $j < i$), and similarly y_i cannot be a link previously broken.

d. $P_i = \sum_{1}^{i} g_j > 0$. (Gain criterion)

e. y_i chosen must permit breaking of an x_{i+1}

f. Before y_i is drawn, check if closing up by joining t_{2i} to t_1 will give a gain value better than the best obtained previously. Let y_i^* be a link connecting t_{2i} to t_1 and let $g_i^* = |x_i| - |y_i^*|$. If $P_{i-1} + g_i^* > P^*$, set $P^* = P_{i-1} + g_i^*$ and let $k = i$. (P^* is now the best improvement in G. $P^* \geq 0$ and must be monotonically nondecreasing. The index k defines the sets to be exchanged to achieve P^*).

4. Terminate the search for x_i and y_i in Steps 1 to 3 when either

a. no further links x_i and y_i satisfy 3(c) to (e).

b. $P_i \leq P^*$

Delete the x_i's and replace them by the y_i's, $i = 1,\ldots,k$, to obtain G'.

5. Test the feasibility of G' (see Sec. 4.2.3). If feasible, take G' as the starting network and repeat from Step 1. Otherwise generate another initial solution using the Starting Routine.

6. If $P^* = 0$, a limited backtracking is invoked:

a. Repeat Steps 3 and 4, choosing y_2's in increasing cost as long as they satisfy the gain criterion $g_1 + g_2 > 0$.

b. If all choices of y_2 in Step 3(b) are exhausted without profit, return to Step 3(a) and try the alternate choice for x_2.

c. If this also fails to give improvement, a further back-up is performed to Step 2, where the y_1's are examined in increasing cost.

d. If the y_1's are also exhausted without profit, try the alternate x_1 in Step 1.

e. If this fails, select another vertex t_1 and repeat from Step 1.

7. The routine terminates when all vertices in G have been designated t_1 without profit. Return to the Starting Routine.

A justification of Step 6 is in order: Experimental results by Lin and Kernighan showed that for the 2-redundancy network, backtracking on the third level and higher, i.e., $i \geq 3$, indicated considerable time penalty, hence the only alternates for x_1, y_1 and x_2,y_2 at the first and second levels are examined. Other refinements to the procedure discussed in the original paper are not applied here since actual computer runs may indicate different time-cost trade-offs for networks with redundancy greater than 2.

4.2.3 Feasibility Test

The optimizing routine for 2-redundancy network has a built-in feasibility test but is not applicable to higher redundancy networks. The fact that G is feasible does not imply that G' is also feasible even though the degree of the vertices satisfies the requirement. It is therefore necessary to test the redundancy after every transformation. Rather than applying Algorithm 2 to examine the new network G', the number of flow calculations can be reduced significantly using the theorems stated by Steiglitz, etal for an X-change or 2-change. Extending this to a k-

change transformation and with uniform redundancy, we need only to check the redundancy between the pairs of vertices whose connecting links were deleted. Algorithm 1 is most suitable for investigating feasibility after every any transformation.

Example:

Given a set of stations whose geographical arrangement is transferred on an Euclidian plane as in Fig. 5.1. We attempt to construct a minimum cost flow network with a uniform redundancy of 3. We model the network as a graph of vertices and branches and illustrate the application of the design procedure.

Start by numbering the vertices uniformly at random as in Fig. 5.1. Measure the distances between vertices (assuming proportional to cost) and obtain the cost matrix. Add branches one at a time and keep track of the number of incident branches on each vertex by using an updated requirement table as shown. Row 1 of this table is filled up by the assigned vertex requirements. This row has all the same entries so start with vertex 1. The cost matrix shows branch (1,6) has the least cost (here the parentheses are used in place of brackets to represent undirected branches). Encircle this entry in the matrix and construct the branch (1,6) in Fig. 5.2. Form row 2 of the requirement table by reducing entries for vertices 1 and 6 by unity. The highest updated requirement in this row is 3 and the highest on the list is vertex 2. From the cost matrix we see that branch (2,7) has the least cost. Encircle this entry and construct the branch (2,7). The procedure is continued and the sequence of branches constructed next are (3,4), (5,8),(1,7),(2,8),(3,6), (4,5),(1,2),(3,7),(4,8), and (5,6).

Cost Matrix

	1	2	3	4	5	6	7	8
1	0	(23)	32	41	43	(14)	(20)	23
2		0	23	20	23	23	(10)	(14)
3			0	(23)	40	(20)	(14)	36
4				0	(23)	36	23	(32)
5					0	(45)	32	(23)
6						0	14	30
7							0	23
8								0

Requirement Table

Vertex List	1	2	3	4	5	6	7	8	Branch
Requirement	3	3	3	3	3	3	3	3	(1,6)
	2	3	3	3	3	2	3	3	(2,7)
	2	2	3	3	3	2	3	3	(3,4)
	2	2	2	2	3	2	2	3	(5,8)
	2	2	2	2	2	2	2	2	(1,7)
	1	2	2	2	2	2	1	2	(2,8)
	1	1	2	2	2	2	1	1	(3,6)
	1	1	1	2	2	1	1	1	(4,5)
	1	1	1	1	1	1	1	1	(1,2)
	0	0	1	1	1	1	1	1	(3,7)
	0	0	0	1	1	1	0	1	(4,8)
	0	0	0	0	1	1	0	0	(5,6)
	0	0	0	0	0	0	0	0	

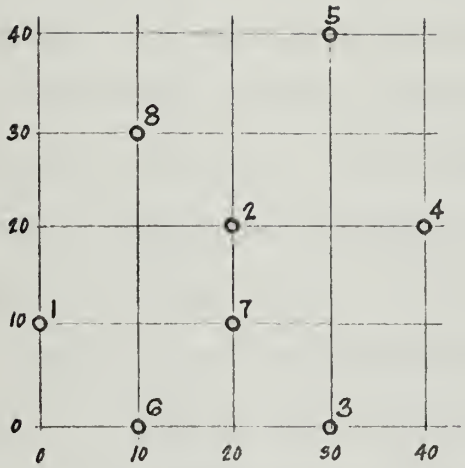


Fig. 5.1

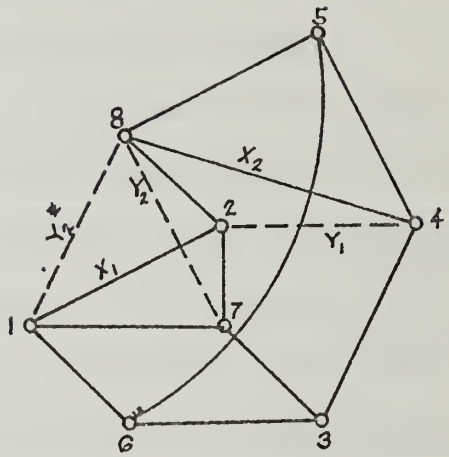


Fig. 5.2

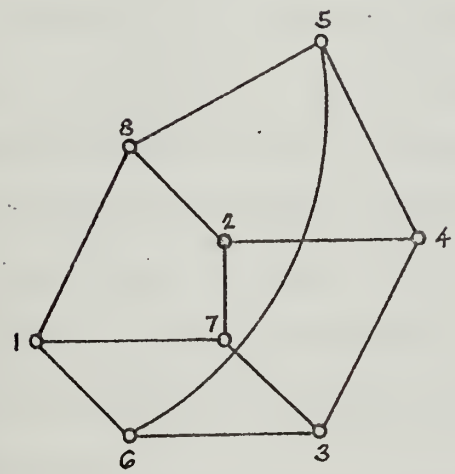


Fig. 5.3

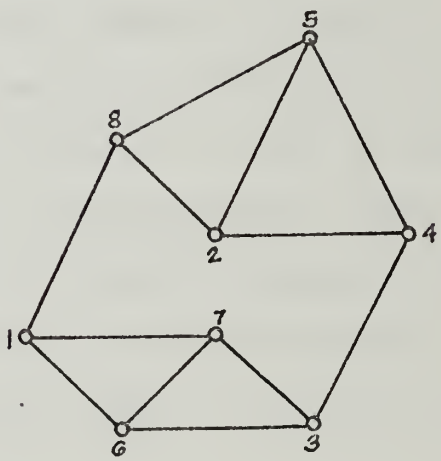


Fig. 5.4

When the number of vertices is odd, the last row of the requirement table has one vertex left with an updated requirement of 1 while the rest are zero. Connect this vertex to one which produces the least branch cost, at the same time satisfying Step 5(c) of the starting routine.

The initial solution generated by the starting routine is shown by solid lines in Fig. 5.2. It is found feasible using Algorithm 2. The cost of the network is obtained by summing up the values of the encircled entries in the cost matrix. For this example, the initial cost is 261.

We apply now the Optimizing Routine to reduce the cost. Set $P^* = 0$ and $i = 1$. In Fig. 5.2, choose vertex 1 as t_1. x_1 is selected such that it is the longest among the three incident branches to t_1. Vertex 2 which is the other endpoint of x_1 is designated t_2. From t_2 find a y_1 using Step 3 of the routine such that $g_1 > 0$. Only the imaginary branch (2,4) is available since the other possibilities (2,5),(2,6), and (2,3) would result to $g_1 \leq 0$. Choosing (2,4) as y_1 the gain $g_1 = |x_1| - |y_1| = 3$. The overall profit $P_1 = 3 > P^*$. Designate vertex 4 as t_3.

Now let $i = 2$. Choose x_2 as indicated. Assign t_4 to vertex 8 which is the endpoint of x_2. Using Step 3 again, y_2 is selected to be the imaginary branch (8,7). The gain $g_2 = 9$ and $P_2 = P_1 + g_2 = 12$. Before going further, check if joining t_4 to t_1 would improve P^*. With (8,1) as y_2^*, $g_2^* = 9$ and $P^* = P_1 + g_2^* = 12$. We have to terminate the search for x_i's and y_i's since $P^* = P_2$ (Step 4). Had P^* been greater than P_2 we could have continued by designating vertex 7 as t_5 and look for x_3 and y_3.

Construct the Y's and delete the X's. The network with improved cost of 249 (not yet optimal) is shown in Fig. 5.3. The cost is obtained by subtracting the costs of x_1 and x_2 from the initial value and then adding the costs of the branches y_1 and y_2^*.

Feasibility of the new network is tested using Algorithm 1 since we need only to check the branch connectivity between the vertex-pairs 1-2 and 4-8. This network is feasible so we can iterate the procedure using Fig. 5.3 as the new starting solution.

Satisfying the vertex requirement does not necessarily mean feasibility is preserved. Fig. 5.4 obtained by the optimizing procedure is not feasible as it has a minimum connectivity of 2.

V. CONCLUSION

Analysis procedures provide convenient tools for evaluating the "strength" or "weakness" of flow networks with respect to some physically meaningful criteria. The computer analysis programs were prepared as implementation of the algorithms developed and are capable of investigating large networks based on either branch or vertex survivability measure.

It was shown that the analysis tools are not only used for measuring performance of existing systems but are also applied effectively in their design particularly when either explicit solutions are impractical or solutions do not appear to be obtainable. A heuristic algorithm for the design of minimum-cost survivable networks is presented. Analysis techniques are found indispensable in the procedure.

Further study is indicated in design perhaps by encoding the algorithm for computer running and to investigate the following:

1. What level of backtracking would be considered optimal?

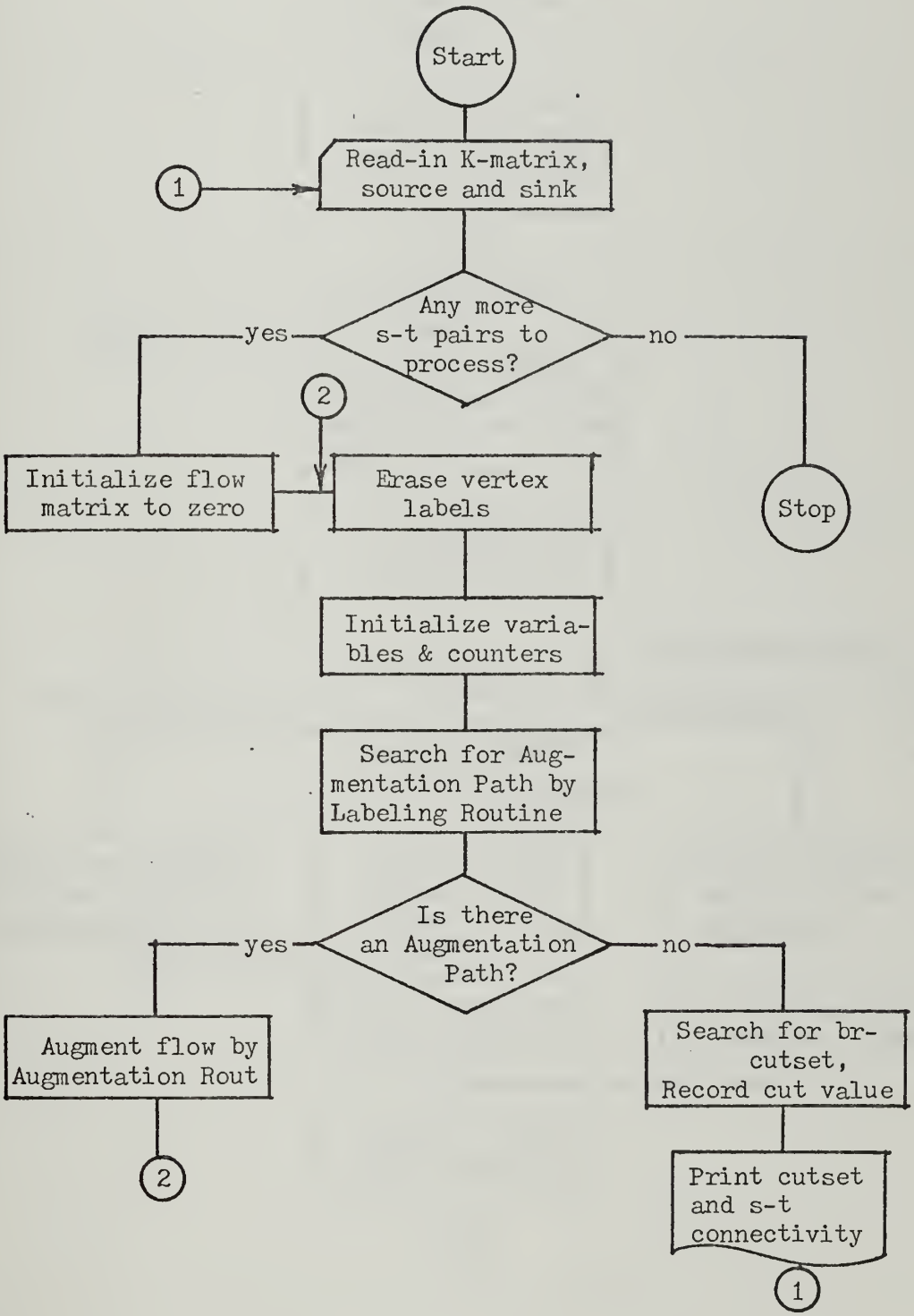
2. How often does infeasibility occur in the starting routine? The probability of occurrence may be different from that obtained by Steiglitz since the criteria are not the same.

3. Are refinements described in [16] applicable to networks with redundancy greater than 2?

APPENDIX A

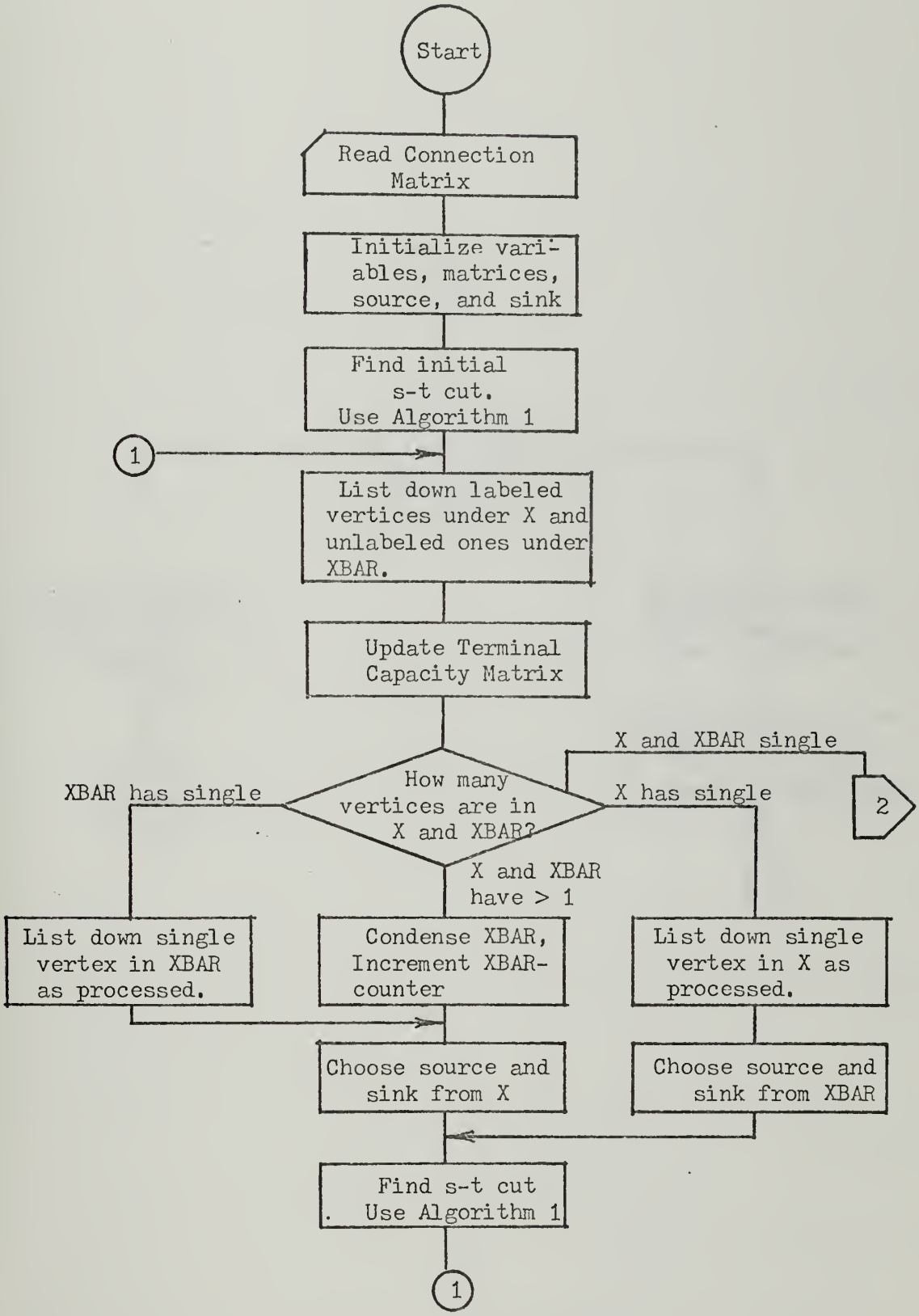
FLOWCHARTS

A-1. s-t BRANCH CONNECTIVITY

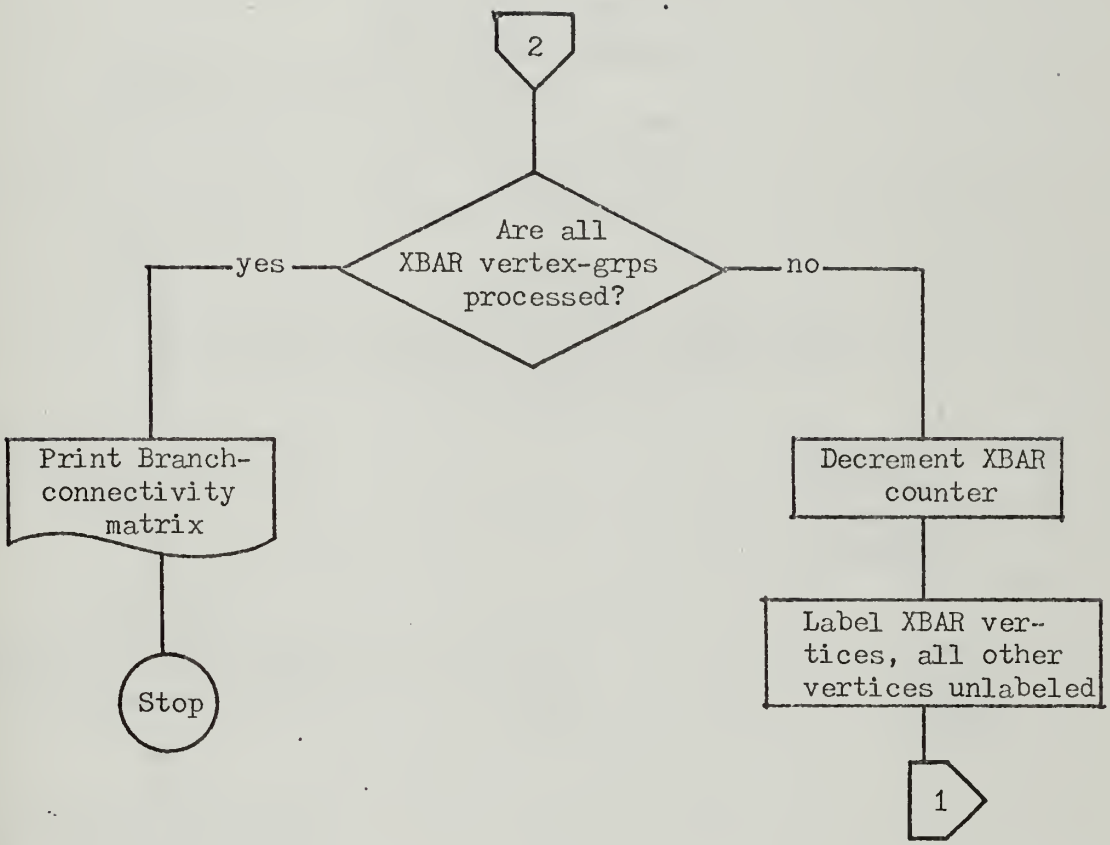


A-2. MULTITERMINAL BRANCH CONNECTIVITY

(Undirected Network)

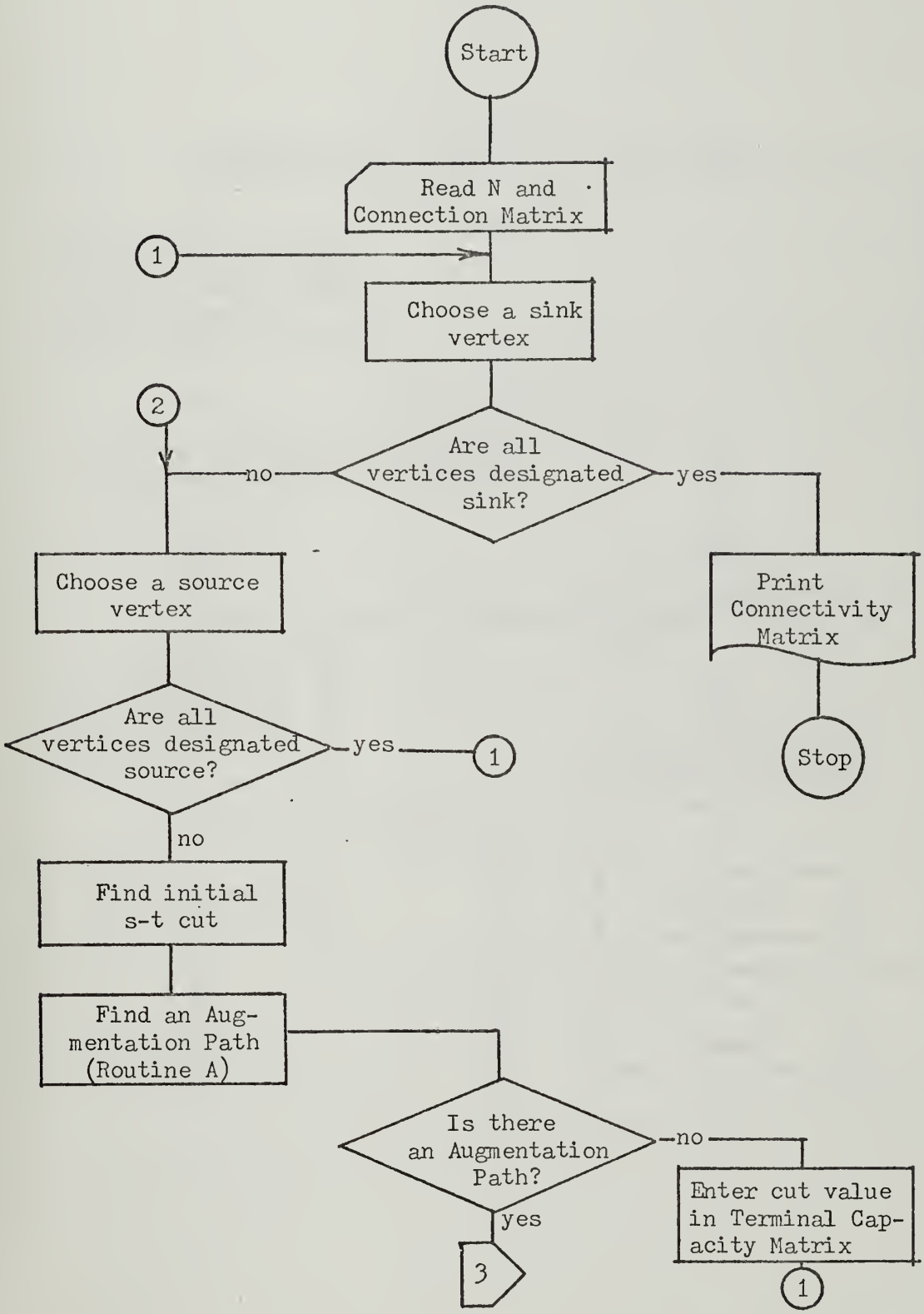


A-2. (continued)

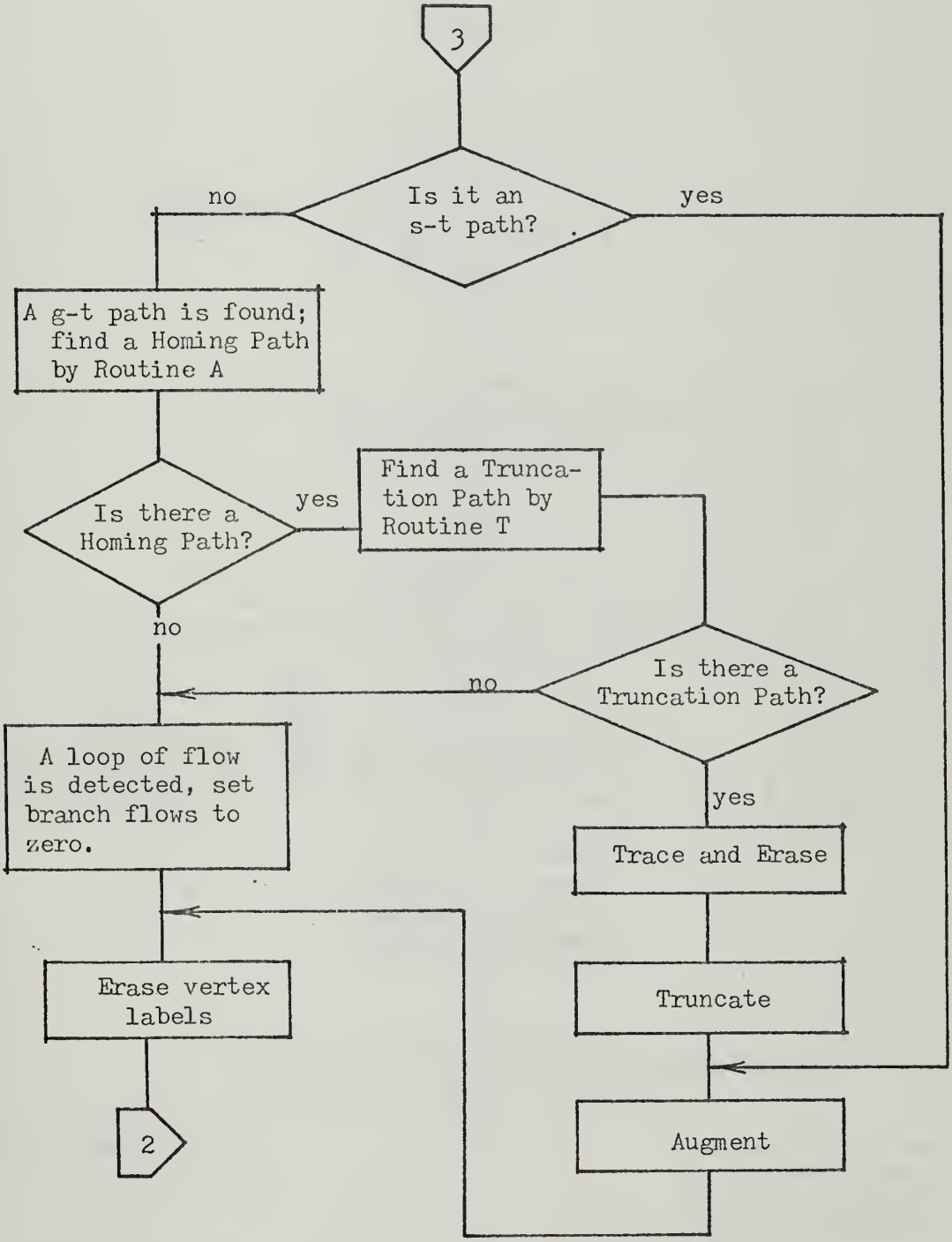


A-3. MULTITERMINAL BRANCH CONNECTIVITY

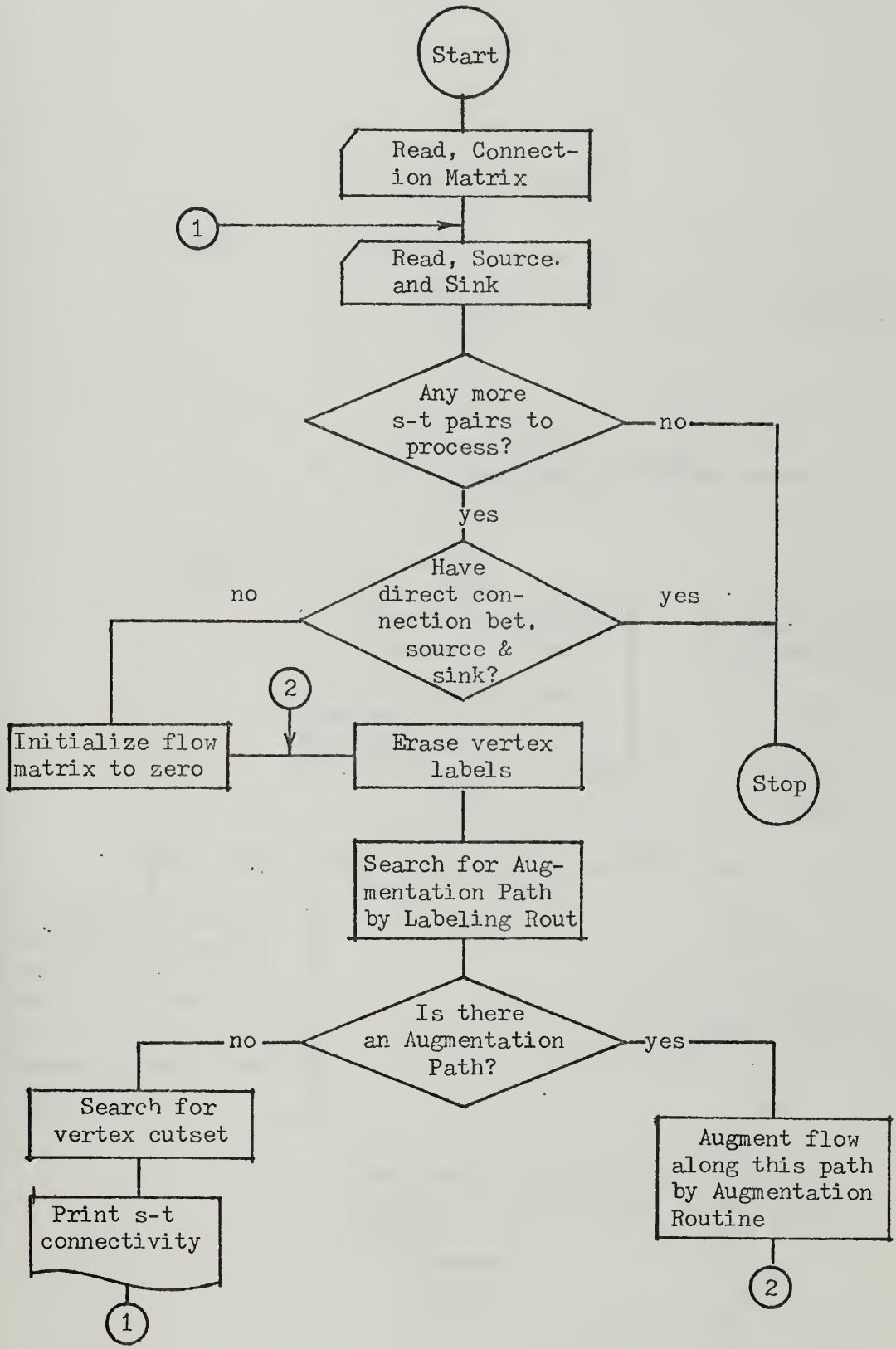
(Directed Network)



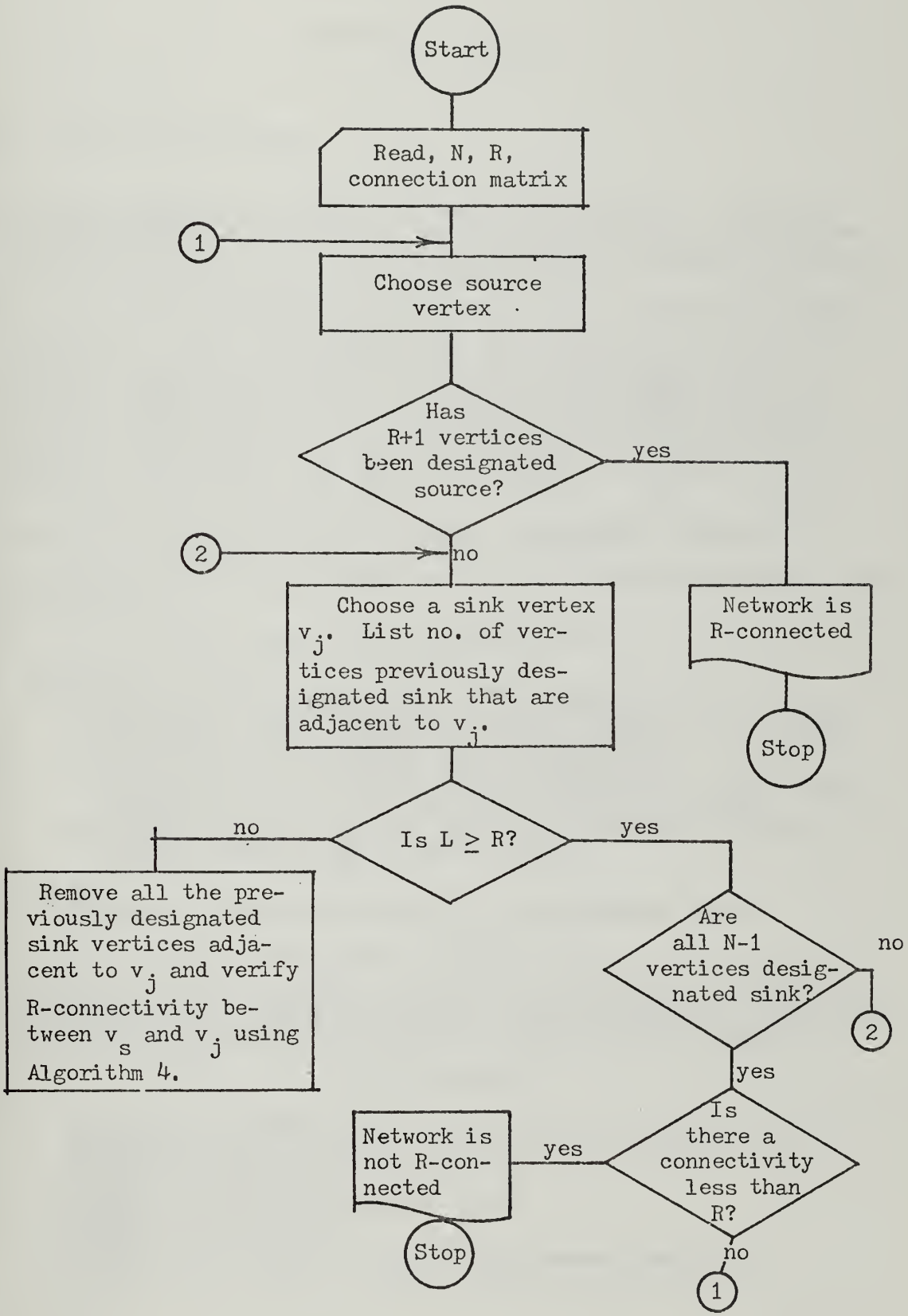
A-3. (continued)



A-4. s-t VERTEX CONNECTIVITY



A-5. UNIFORM R- CONNECTIVITY



APPENDIX B

COMPUTER PROGRAMS

B-1. PROGRAM MAXCON-1

```
C    THIS PROGRAM COMPUTES FOR THE MULTITERMINAL  BRANCH-
C CONNECTIVITY OF A DIRECTED OR MIXED NETWORK. THE NETWORK
C IS REPRESENTED BY ITS CONNECTION MATRIX FOR COMPUTATION.
C THE NETWORK MUST HAVE ONLY SINGLE BRANCH BETWEEN VERTICES.
C
C    THE PROGRAM PROCEEDS BY READING-IN THE INPUT DATA CON-
C SISTING OF THE TOTAL NUMBER OF VERTICES AND THE CONNECT-
C ION MATRIX. IT THEN DETERMINES THE INITIAL S'-T CUT WITH
C THE FIRST AND NTH VERTICES AS SOURCE AND SINK RESP.
C HAVING FOUND AN S'-T CUT, IT THEN FINDS OTHER S-T CUTS
C KEEPING THE SINK VERTEX CONSTANT AND VARYING ONLY THE
C SOURCE VERTEX FROM 2 TO N. AFTER COMPLETING THIS LOOP,
C THE SINK VERTEX IS SET TO N-1 AND THEN FINDS AN S'-T CUT
C BETWEEN VERTICES 1 AND N-1. THE SOURCE IS AGAIN VARIED
C FROM 2 TO N-1 FOR THE OTHER S-T CUTS.
C
C    THE PROCESS IS REPEATED UNTIL N(N-1) CUTS ARE DETER-
C MINED. EACH REPETITION INVOLVES SUBROUTINE FLOVAR.
C
C    THE OUTPUT IS A PRINTOUT OF THE NXN BRANCH-CONNECTIVITY
C MATRIX.
C
C . . . . . IDENTIFIER NOMENCLATURE . . . . . .
C
C  ARRAYS
C     KMAT--- INPUT CONNECTION MATRIX WITH ENTRIES 0, 1, -1
C     DMAT--- S'-T FLOW MATRIX REPRESENTING AN S'-T FLOW
C        PATTERN. THIS MATRIX IS UPDATED BEFORE A NEW SET
C        OF S AND S' VERTICES ARE DESIGNATED.
C     FLOMAT- S-T FLOW MATRIX REPRESENTING S-T FLOW PAT-
C        TERN RESULTING FROM AUGMENTATION AND HOMING STEPS.
C        NONZERO ENTRIES OF THIS MATRIX UPDATE DMAT.
C     TERMAT- MATRIX DENOTING THE MAXIMUM BRANCH-CONNECT-
C        IVITY BETWEEN ANY VERTEX-PAIR.
C
C  VECTOR
C     NODE--- LIST OF VERTICES OF THE NETWORK GRAPH.
C
C . . . . . ADDITIONAL INFORMATION . . . . . .
C
C    THIS PROGRAM IS CODED IN FORTRAN LANGUAGE. NETWORK OF
C ANY SIZE CAN BE HANDLED BY THIS PROGRAM WITHIN THE LIMIT-
C ATIONS OF THE COMPUTER STORAGE. THE DIMENSION, READ, AND
C WRITE STATEMENTS MUST HOWEVER BE ADJUSTED ACCORDINGLY.
C
C . . . . . MAIN PROGRAM . . . . . .
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(25,25), FLOMAT(25,25), DMAT(25,25)
      DIMENSION TERMAT(25,25)
      DIMENSION NODE(25)
      COMMON /ARR1/ KMAT,FLOMAT,DMAT
      COMMON /ARR2/ TERMAT
      COMMON /CH/ NODE
      COMMON /PT/ SOURCE,SPRIME,SINK,AMARK,BRCUT,N,G
C
CCC
C
```

```
      READ (5,9) N
      READ (5,10) ((KMAT(I,J),J=1,25),I=1,N)
C          INITIALIZE TERMINAL MATRIX
C
      DO 2 X=1,N
C
      DO 1 Y=1,25
      TERMAT(X,Y)=0
    1 CONTINUE
C
    2 CONTINUE
C
C
C          COMPUTE FOR THE N(N-1) BRANCH CUTS
C
      DO 8 K=1,N
      J=N+1-K
C
C          SET TO ZERO THE FLOW MATRICES
C
      DO 4 L=1,N
C
      DO 3 M=1,N
      DMAT(L,M)=0
      FLOMAT(L,M)=0
    3 CONTINUE
C
    4 CONTINUE
C
      SPRIME=0
C
      DO 7 I=1,N
      IF (I.EQ.J) GO TO 7
      SOURCE=I
      SINK=J
      BRCUT=0
      CALL FLOVAR
C
C          ENTER CUT VALUE INTO THE TERMINAL MATRIX
      TERMAT(I,J)=BRCUT
      SPRIME=I
C
C          TRANSFER FLOMAT ENTRIES INTO DMAT
C
      DO 6 L=1,N
C
      DO 5 M=1,N
      IF (FLOMAT(L,M).EQ.0) GO TO 5
      DMAT(L,M)=FLOMAT(L,M)
C
C          RESET FLOMAT TO ZERO
      FLOMAT(L,M)=0
    5 CONTINUE
C
    6 CONTINUE
C
    7 CONTINUE
C
    8 CONTINUE
C
C
C          COMPUTATION COMPLETED
      WRITE (6,11)
      WRITE (6,12)
      CALL PRTMAT
      STOP
C
    9 FORMAT (I6)
   10 FORMAT (25I3)
   11 FORMAT ('0',T14,'BRANCH-CONNECTIVITY MATRIX')
   12 FORMAT (T14,'--------------------------')
      END
```

```
C . . . . . SUBROUTINE FLOVAR . . . . . .
C
C   THIS SUBROUTINE DETERMINES AN S-T CUT ONCE AN INITIAL
C S'-T CUT IS FOUND. IT MAXIMIZES AN S-T FLOW USING THE
C EXISTING S'-T FLOW PATTERN.
C
      SUBROUTINE FLOVAR
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(25,25), FLOMAT(25,25), DMAT(25,25)
      DIMENSION NODE(25)
      COMMON /ARR1/ KMAT,FLOMAT,DMAT
      COMMON /CH/ NODE
      COMMON /PT/ SOURCE,SPRIME,SINK,AMARK,BRCUT,N,G
C
C          ERASE ALL VERTEX LABELS
C
    1 DO 2 J=1,N
      NODE(J)=0
    2 CONTINUE
C
C
C          SEARCH FOR AUGMENTATION PATH
      CALL AUGMNT
      IF (AMARK) 3,24,15
C
C          AN S-G AUGMENTATION PATH IS FOUND
C          SEARCH FOR HOMING PATH
    3 Q=G
C
    4 DO 5 J=1,N
      IF (DMAT(Q,J).EQ.1) GO TO 6
    5 CONTINUE
C
      GO TO 1
    6 NODE(J)=Q
      Q=J
      IF (Q.EQ.G) GO TO 19
      IF (J.EQ.SINK) GO TO 7
      GO TO 4
C
C          A HOMING PATH IS FOUND
C          SEARCH FOR TRUNCATION PATH
    7 Z=G
      IF (G.EQ.SPRIME) GO TO 12
C
    8 DO 9 I=1,N
      IF (DMAT(I,Z).EQ.1) GO TO 10
    9 CONTINUE
C
      IF (Z.EQ.G) GO TO 12
      GO TO 11
   10 NODE(I)=Z
      Z=I
      IF (Z.EQ.G) GO TO 21
      IF (Z.EQ.SPRIME) GO TO 11
      GO TO 8
C
C          A TRUNCATION PATH IS FOUND
C          TRUNCATE SPRIME-G PATH
   11 I=NODE(Z)
      DMAT(Z,I)=0
      Z=I
      IF (Z.EQ.G) GO TO 12
      GO TO 11
C
C          TRACE AND ERASE G-T PATH
   12 Q=SINK
   13 J=NODE(Q)
      FLOMAT(J,Q)=1
      DMAT(J,Q)=0
      Q=J
      IF (J.EQ.G) GO TO 14
```

```
      GO TO 13
C
C          AUGMENT S-G PATH
   14 IF (G.EQ.SOURCE) GO TO 23
      J=G
      GO TO 16
C
C          AUGMENT S-T PATH
   15 J=SINK
   16 I=NODE(J)
      IF (I.LT.0) GO TO 17
C
C          INCREASE FORWARD FLOW BY UNITY
      FLOMAT(I,J)=1
      GO TO 18
C
C          DECREASE BACKFLOW BY UNITY.
   17 I=-I
      FLOMAT(J,I)=0
   18 J=I
      IF (J.EQ.SOURCE) GO TO 23
      GO TO 16
C
C          A LOOP OF FLOW IS FOUND BY HOMING ROUTINE
   19 Q=G
   20 J=NODE(Q)
      DMAT(J,Q)=0
      Q=J
      IF (Q.EQ.G) GO TO 1
      GO TO 20
C
C          A LOOP OF FLOW IS FOUND BY TRUNCATION ROUTINE
   21 Z=G
   22 I=NODE(Z)
      DMAT(Z,I)=0
      Z=I
      IF (Z.EQ.G) GO TO 1
      GO TO 22
   23 BRCUT=BRCUT+1
      GO TO 1
C
C          NO MORE AUGMENTATION PATH IS FOUND. FLOW PATTERN
C            IS MAX
   24 RETURN
      END
C
C
C
C . . . . . SUBROUTINE AUGMNT . . . . . .
C
C    THIS SUBROUTINE IS USED TO DETECT AN S-G OR S-T PATH
C USING THE MODIFIED FORD AND FULKERSON LABELING ALGORITHM
C
      SUBROUTINE AUGMNT
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(25,25), FLOMAT(25,25), DMAT(25,25)
      DIMENSION NODE(25), JSTORE(10), ISTORE(10)
      COMMON /ARR1/ KMAT,FLOMAT,DMAT
      COMMON /CH/ NODE
      COMMON /PT/ SOURCE,SPRIME,SINK,AMARK,BRCUT,N,G
C
      I=SOURCE
      L=0
      M=0
      Z=0
      AMARK=0
      NODE(I)=SOURCE
C
C          SEARCH FOR AUGMENTING PATH
C
    1 DO 4 J=1,N
C
```

```
C          TEST FOR CONNECTING BRANCH BETWEEN VERTEX PAIRS
      IF (DMAT(J,I).EQ.0) GO TO 2
      IF (J.EQ.SOURCE) GO TO 4
C
C          A POSITIVE BACKFLOW IS ENCOUNTERED
      IF (NODE(J).NE.0) GO TO 4
C
C          LABEL VERTEX J
      NODE(J)=-I
      GO TO 3
    2 IF (KMAT(I,J).NE.1) GO TO 4
C
C          A FORWARD BRANCH IS ENCOUNTERED. TEST FOR FORWARD
C            FLOW
      IF (DMAT(I,J).NE.0) GO TO 7
      IF (FLOMAT(I,J).NE.0) GO TO 4
      IF (NODE(J).NE.0) GO TO 4
C
C          LABEL VERTEX J
      NODE(J)=I
C
C          IS THE SINK VERTEX REACHED?
    3 IF (J.EQ.SINK) GO TO 8
      M=M+1
      JSTORE(M)=J
    4 CONTINUE
C
      IF (Z.NE.0) GO TO 6
      IF (M.EQ.0) GO TO 9
C
C          TRANSFER CONTENTS OF JSTORE TO ISTORE
C
      DO 5 S=1,M
      ISTORE(S)=JSTORE(S)
    5 CONTINUE
C
      Z=M
      M=0
    6 I=ISTORE(Z)
      Z=Z-1
      GO TO 1
C
C          AN S-G AUGMENTATION PATH IS FOUND
    7 G=I
      AMARK=-1
      GO TO 10
C
C          AN S-T AUGMENTATION PATH IS FOUND
    8 AMARK=1
      GO TO 10
C
C          NO AUGMENTATION PATH FOUND
    9 AMARK=0
   10 RETURN
      END
C
C
C
C . . . . . SUBROUTINE PRTMAT . . . . . .
C
C     THIS SUBROUTINE PRINTS TERMAT IN PROPER FORM AND IN
C FORMAT 35I3.
C
C   IF ARRAY LENGTH IS .LE. 35 A SINGLE LOOP SECTION PRINTS
C THE MATRIX, OTHERWISE IT PRINTS THE FIRST 35 COLUMNS FOL-
C LOWED BY THE NEXT 35 COLUMNS AND SO ON IN MULTIPLES OF 35
C USING MULTIPLE LOOPS. THE REMAINDER COLUMNS ARE PRINTED
C IMMEDIATELY BELOW THE LAST 35-COLUMN GROUP.
C
C
C
C
```

```
      SUBROUTINE PRTMAT
      IMPLICIT INTEGER(A-Z)
      DIMENSION TERMAT(25,25)
      COMMON /ARR2/ TERMAT
      COMMON /PT/ SOURCE,SPRIME,SINK,AMARK,BRCUT,N,G
C
C         HOW MANY MULTIPLES OF 35 IS N?
      MULPLE=N/35
      IF (MULPLE.GE.1) GO TO 2
C
C         THIS SECTION PRINTS ARRAY WITH LENGTH .LT. 35
C
      DO 1 I=1,N
      WRITE (6,8) I,(TERMAT(I,J),J=1,N)
    1 CONTINUE
C
      GO TO 7
C
C         THIS SECTION PRINTS ARRAY WITH LENGTH .GE. 35
C           IN GROUPS OF 35
    2 M=1
      K=1
    3 MPLS35=M+35
C
      DO 4 I=1,N
      WRITE (6,8) I,(TERMAT(I,J),J=M,MPLS35)
    4 CONTINUE
C
      M=M+35
      IF (K.EQ.MULPLE) GO TO 5
      WRITE (6,9) K
      WRITE (6,10)
      K=K+1
      GO TO 3
C
C         THIS SECTION PRINTS REMAINDER COLUMNS
    5 REMDR=N-MULPLE*35
      IF (REMDR.LE.0) GO TO 7
      WRITE (6,11)
      WRITE (6,12)
C
      DO 6 L=1,N
      WRITE (6,8) L,(TERMAT(L,J),J=M,N)
    6 CONTINUE
C
    7 RETURN
C
    8 FORMAT ('0', T8, I2, '/', 1X, 35I3)
    9 FORMAT (//T14,'CONTINUATION', I3, 2X, 'OF TERMAT')
   10 FORMAT (T14,'--------------------------')
   11 FORMAT (//T14,'LAST SECTION OF TERMAT')
   12 FORMAT (T14,'-------------------------')
      END
```

B-2. PROGRAM MAXCON-1A

```
C     THIS PROGRAM COMPUTES FOR THE BRANCH-CONNECTIVITY BET-
C  WEEN ANY TWO GIVEN VERTICES OF A DIRECTED OR MIXED NET-
C  WORK. THE NETWORK GRAPH IS REPRESENTED BY ITS CONNECTION
C  MATRIX FOR COMPUTATION. NETWORK MUST HAVE ONLY SINGLE
C  BRANCH BETWEEN VERTICES.
C
C     THE NUMBER OF VERTICES, THE SELECTED SOURCE AND SINK
C  VERTICES, AND THE CONNECTION MATRIX ARE THE INPUTS. THE
C  PROGRAM DOES A MAXFLOW CALCULATION USING THE MODIFIED
C  FORD AND FULKERSON LABELING ALGORITHM. COMPUTED CUT VALUE
C  IS THE NETWORK CONNECTIVITY WITH RESPECT TO THE SPECIFIED
C  VERTEX-PAIR.
C
C     SEVERAL VERTEX-PAIRS MAYBE LISTED WITH THE INPUT AND
C  THE PROGRAM DOES SIMILAR MAXFLOW CALCULATION FOR EACH
C  PAIR.
C
C     THE OUTPUT IS A PRINTOUT CONSISTING OF THE GIVEN SOURCE
C  AND SINK, THE CONNECTIVITY VALUE, AND THE BRANCH-CUTSET
C  AS DENOTED BY THE VERTICES TO WHICH EACH BRANCH CUT IS
C  CONNECTED.
C
C     THE PRESENT DIMENSION AND READ STATEMENTS MAYBE ADJUST-
C  ED ACCORDING TO THE SIZE OF THE NETWORK TO BE ANALYZED.
C
C     THIS PROGRAM MAYBE PREFERRED OVER MAXCON-1 WHEN ONLY A
C  FEW SOURCE-SINK PAIRS ARE TO BE ANALYZED.
C
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(50,50), FLOMAT(50,50), NODE(50), NX(50)
      DIMENSION NOTNX(20), JSTORE(20), ISTORE(20)
C
      READ (5,22) N
      READ (5,23) ((KMAT(I,J),J=1,50),I=1,N)
      WRITE (6,25)
    1 READ (5,24) SOURCE,SINK
      IF (SOURCE.EQ.0) GO TO 21
C
C          INITIALIZE FLOW MATRIX TO ZERO
C
      DO 3 I=1,N
C
      DO 2 J=1,N
      FLOMAT(I,J)=0
    2 CONTINUE
C
    3 CONTINUE
C
C
C          ERASE VERTEX LABELS
C
    4 DO 5 J=1,N
      NODE(J)=0
    5 CONTINUE
C
C
      I=SOURCE
      L=0
      M=0
      Z=0
      BRCUT=0
      NODE(I)=SOURCE
C
C          SEARCH FOR AUGMENTING PATH
C
    6 DO 10 J=1,N
C
```

```
C          TEST FOR CONNECTING BRANCH BETWEEN VERTEX-PAIR
      IF (KMAT(I,J)) 7,10,8
C
C          A BACKWARD BRANCH IS ENCOUNTERED. TEST FOR POSI-
C            TIVE BACKFLOW.
    7 IF (FLOMAT(J,I).EQ.0) GO TO 10
      IF (NODE(J).NE.0) GO TO 10
C
C          LABEL NODE(J)
      NODE(J)=-I
      GO TO 9
C
C          A FORWARD BRANCH IS ENCOUNTERED. TEST FOR ZERO
C            FORWARD FLOW.
    8 IF (FLOMAT(I,J).NE.0) GO TO 10
C
C          IS NODE(J) LABELED?
      IF (NODE(J).NE.0) GO TO 10
      NODE(J)=I
C
C          IS THIS THE SINK VERTEX?
    9 IF (J.EQ.SINK) GO TO 13
      M=M+1
      JSTORE(M)=J
   10 CONTINUE
C
      IF (Z.NE.0) GO TO 12
      IF (M.EQ.0) GO TO 16
C
C          TRANSFER CONTENTS OF JSTORE TO ISTORE
C
      DO 11 S=1,M
      ISTORE(S)=JSTORE(S)
   11 CONTINUE
C
      Z=M
      M=0
   12 I=ISTORE(Z)
      Z=Z-1
      GO TO 6
C
C          WE REACH HERE IF AN AUGMENTING PATH IS FOUND
C            ADJUST FLOW ON ALL BRANCHES ALONG THIS PATH.
   13 I=NODE(J)
      IF (I.LT.0) GO TO 14
C
C          INCREASE FORWARD FLOW BY UNITY
      FLOMAT(I,J)=1
      GO TO 15
C
C          DECREASE BACKFLOW BY UNITY
   14 I=-I
      FLOMAT(J,I)=0
   15 J=I
      IF (J.EQ.SOURCE) GO TO 4
      GO TO 13
C
C          THIS STEP IS REACHED IF LABELING AND FLOW AUGMENT-
C            ATION IS COMPLETED. SEARCH FOR BRANCH CUTS.
C
   16 DO 19 I=1,N
      IF (NODE(I).EQ.0) GO TO 19
C
      DO 18 J=1,N
      ADD=1
      IF (KMAT(I,J).EQ.0) GO TO 18
      IF (NODE(J).NE.0) GO TO 18
C
C          IF FLOW OCCURS IN BOTH DIRECTIONS, NET FLOW IS ZERO
      IF (FLOMAT(I,J).EQ.FLOMAT(J,I)) GO TO 18
      IF (FLOMAT(I,J).EQ.1) GO TO 17
      ADD=FLOMAT(I,J)
```

```
   17 BRCUT=BRCUT+ADD
      L=L+1
      NX(L)=I
      NOTNX(L)=J
   18 CONTINUE
C
   19 CONTINUE
C
C
C          COMPUTATION TERMINATES SINCE NO MORE AUGMENTING
C            PATH IS FOUND.
      IF (BRCUT.NE.0) GO TO 20
      WRITE (6,26) SOURCE,SINK,BRCUT
      GO TO 1
   20 WRITE (6,27) SOURCE,SINK,BRCUT,(NX(J),NOTNX(J),J=1,L)
      GO TO 1
   21 STOP
C
   22 FORMAT (I6)
   23 FORMAT (25I3,/25I3)
   24 FORMAT (2I6)
   25 FORMAT ('0', 10X, 'SOURCE', 10X, 'SINK', 10X, 'BR-
     1     CONNECTIVITY', 10X, 'BR-CUTSET')
   26 FORMAT ('0', 12X, I2, 13X, I2, 17X, I2)
   27 FORMAT ('0', 12X, I2, 13X, I2, 17X, I2,
     1     (T67, I3, 1X, '--', I3/))
      END
```

B-3. PROGRAM MAXCON-2

```
C     THIS PROGRAM COMPUTES FOR THE MULTITERMINAL BRANCH-
C CONNECTIVITY OF ANY GIVEN UNDIRECTED NETWORK. THE NETWORK
C GRAPH IS PRESENTED FOR COMPUTATION BY ITS CONNECTION MAT-
C RIX. NETWORK CAN HAVE PARALLEL BRANCHES BETWEEN VERTICES.
C
C     THE PROGRAM PROCEEDS BY FIRST READING-IN THE NUMBER OF
C VERTICES, THE INITIAL VERTEX-PAIR CHOSEN ARBITRARILY,
C AND THE CONNECTION MATRIX. THE CUT BETWEEN THE TWO
C SELECTED VERTICES IS OBTAINED USING SUBROUTINE MINCUT.
C THE CONNECTIVITY FOR THE CUT(X,XBAR) IS THE COMPUTED CUT
C VALUE. THIS IS ENTERED INTO THE TERMINAL CAPACITY MATRIX
C BY SUBROUTINE MATRXT. THE SUCCEEDING VERTEX-PAIRS ARE
C SELECTED BY THE PROGRAM FROM THE GENERALIZED X-VERTEX-
C GROUP. SUBROUTINE MINCUT IS EMPLOYED EACH TIME FOR
C OBTAINING THE CUT VALUE. WHEN ALL VERTICES IN X ARE EX-
C HAUSTED, THE X-VERTEX-GROUP IS CONDENSED INTO A SINGLE
C VERTEX BY SUBROUTINE MATRXC AND THE XBAR-VERTEX-GROUP IS
C IN TURN PROCESSED SIMILARLY. FOR EACH COMPUTATION, SUB-
C ROUTINE MATRXT UPDATES THE TERMINAL CAPACITY MATRIX.
C
C    WITH N AS THE TOTAL NUMBER OF VERTICES, THE PROGRAM
C WILL DO N-1 SUCH COMPUTATIONS AND FINALLY PRESENT A MULTI-
C TERMINAL BRANCH-CONNECTIVITY MATRIX AS OUTPUT.
C
C . . . . . IDENTIFIER NOMENCLATURE . . . . . .
C
C  ARRAYS
C     KMAT----INPUT CONNECTION MATRIX WHICH HAS ENTRIES
C             GREATER THAN UNITY INDICATING PARALLEL BRAN-
C             CHES BETWEEN VERTICES
C     CMAT----MATRIX REPRESENTING THE NEW GRAPH FORMED RES-
C             ULTING FROM VERTEX CONDENSATION. IT HAS ALSO
C             ENTRIES GREATER THAN UNITY.
C     FLOMAT--MATRIX INDICATING MAXIMUM FLOW THAT CAN EXIST
C             BETWEEN ANY VERTEX-PAIR ALLOWING UNIT CAPA-
C             CITY FOR EACH BRANCH.
C     TERMAT--MATRIX DENOTING THE MAXIMUM CONNECTIVITY BET-
C             WEEN ANY VERTEX-PAIR.
C     VRTX   )
C     NOTVTX)-ARRAY STORAGE FOR SEQUENCE OF CUTS(X,XBAR),
C             HAVING MORE THAN ONE VERTEX ON EACH SIDE.
C
C  VECTORS
C     NX------VECTOR STORAGE FOR X-VERTICES IN CUT(X,XBAR)
C             CURRENTLY BEING PROCESSED.
C     NOTNX---VECTOR STORAGE FOR XBAR-VERTICES IN CUT
C             (X,XBAR) CURRENTLY BEING PROCESSED.
C     NLIST---STORAGE FOR VERTICES WHICH HAVE OCCURED SING-
C             ULAR IN A CUT(X,XBAR).
C
C  CONSTANTS AND COUNTERS
C     N-------TOTAL NUMBER OF VERTICES IN THE GRAPH
C     SOURCE--CHOSEN SOURCE VERTEX
C     SINK----CHOSEN SINK VERTEX
C     ROW-----DESIGNATES ROW NUMBER IN ARRAYS VRTX AND
C             NOTVTX
C     BRCUT---DENOTES CUT VALUE
C     COUNT1--NUMBER OF VERTICES IN NX
C     COUNT2--NUMBER OF VERTICES IN NOTNX
C     NCOUNT--RUNNING TOTAL OF VERTICES IN NLIST
C
C . . . . . ADDITIONAL INFORMATION . . . . . .
C
C     THIS PROGRAM IS CODED IN FORTRAN LANGUAGE. THE PRESENT
C DIMENSION STATEMENTS REQUIRE 150K OF COMPUTER STORAGE.
C LARGER NETWORKS CAN BE HANDLED BY ADJUSTING THE DIMENSION,
C READ, AND WRITE STATEMENTS.
C
```

```
C . . . . . MAIN PROGRAM . . . . . .
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(35,35), FLOMAT(35,35), CMAT(35,35)
      DIMENSION TERMAT(35,35)
      DIMENSION VRTX(35,35), NOTVTX(35,35), NODE(35)
      DIMENSION NX(35), NOTNX(35)
      DIMENSION NLIST(35)
      COMMON /ARR1/ KMAT,FLOMAT,CMAT,VRTX,NOTVTX
      COMMON /ARR2/ TERMAT
      COMMON /CH1/ NODE,NLIST
      COMMON /CH2/ NX,NOTNX
      COMMON /PT1/ N,SOURCE,SINK,ROW,MARK,FLAG,BRCUT
      COMMON /PT2/ COUNT1,COUNT2,NCOUNT
C
      READ (5,29) N,SOURCE,SINK
      READ (5,30) ((KMAT(I,J),J=1,35),I=1,N)
C
C         INITIALIZE VARIABLES
C
      DO 3 I=1,N
C
      DO 2 K=1,10
C
      DO 1 J=1,N
      CMAT(I,J)=KMAT(I,J)
      VRTX(I,J)=0
      NOTVTX(I,J)=0
      NLIST(J)=0
    1 CONTINUE
C
    2 CONTINUE
C
    3 CONTINUE
C
C
      DO 5 I=1,35
C
      DO 4 J=1,35
      TERMAT(I,J)=0
    4 CONTINUE
C
    5 CONTINUE
C
      BRCUT=0
      NCOUNT=0
      FLAG=0
      MARK=0
      ROW=0
C
C         FIND INITIAL S-T CUT
    6 CALL MINCUT
C
C         LIST DOWN LABELED VERTICES UNDER NX AND UNLABELED
C           ONES UNDER NOTNX
      COUNT1=0
      COUNT2=0
C
      DO 7 J=1,N
      NX(J)=0
      NOTNX(J)=0
    7 CONTINUE
C
C
      DO 14 I=1,N
C
C         TEST IF THIS VERTEX IS ALREADY IN NLIST
      IF (NCOUNT.EQ.0) GO TO 9
C
      DO 8 JJ=1,NCOUNT
      IF (NLIST(JJ).EQ.I) GO TO 14
    8 CONTINUE
```

```
C
C
C           TEST IF IT BELONGS TO SOME CONDENSED NOTVTX
    9 ISTORE=ROW
   10 IF (ISTORE.EQ.0) GO TO 12
C
      DO 11 KK=1,N
      IF (NOTVTX(ISTORE,KK).EQ.I) GO TO 14
   11 CONTINUE
C
      ISTORE=ISTORE-1
      GO TO 10
   12 IF (NODE(I).EQ.0) GO TO 13
      COUNT1=COUNT1+1
      NX(COUNT1)=I
      GO TO 14
   13 COUNT2=COUNT2+1
      NOTNX(COUNT2)=I
   14 CONTINUE
C
C
C           UPDATE TERMINAL CAPACITY MATRIX ENTRIES
      CALL MATRXT
C
      IF (COUNT1.EQ.1) GO TO 18
      IF (COUNT2.EQ.1) GO TO 17
C
C           BOTH NX AND NOTNX HAVE MORE THAN ONE VERTEX
      MARK=0
      ROW=ROW+1
C
C           STORE NX AND NOTNX IN VRTX AND NOTVTX RESP
C
      DO 15 LL=1,COUNT1
      VRTX(ROW,LL)=NX(LL)
   15 CONTINUE
C
C
      DO 16 LL=1,COUNT2
      NOTVTX(ROW,LL)=NOTNX(LL)
   16 CONTINUE
C
C
C           CONDENSE NOTVTX INTO SINGLE NODE
      CALL MATRXC
C
C           SELECT TWO VERTICES IN NX AS SOURCE AND SINK
      SOURCE=NX(1)
      SINK=NX(COUNT1)
      GO TO 6
C
C           NX HAS MORE THAN ONE VERTEX AND NOTNX HAS ONE
   17 NCOUNT=NCOUNT+1
      NLIST(NCOUNT)=NOTNX(1)
      SOURCE=NX(1)
      SINK=NX(COUNT1)
      GO TO 6
C
C           NX HAS SINGLE VERTEX AND NOTNX HAS MORE THAN ONE
   18 NCOUNT=NCOUNT+1
      NLIST(NCOUNT)=NX(1)
      IF (COUNT2.EQ.1) GO TO 19
      SOURCE=NOTNX(1)
      SINK=NOTNX(COUNT2)
      GO TO 6
C
C           BOTH NX AND NOTNX HAVE SINGLE VERTEX
C
   19 DO 20 LL=1,NCOUNT
      IF (NLIST(LL).EQ.NOTNX(1)) GO TO 21
   20 CONTINUE
C
```

```
      NCOUNT=NCOUNT+1
      NLIST(NCOUNT)=NOTNX(1)
C
C          TEST IF ALL VERTICES ARE IN NLIST
   21 IF (NCOUNT.EQ.N) GO TO 28
C
C          THIS SECTION PROCESSES THOSE NOTVTX GROUPS
C           PREVIOUSLY CONDENSED
      CCUNT1=0
      CCUNT2=0
C
      DO 23 MM=1,N
      IF (NOTVTX(ROW,MM).EQ.0) GO TO 22
      CCUNT2=CCUNT2+1
   22 IF (VRTX(ROW,MM).EQ.0) GO TO 23
      CCUNT1=CCUNT1+1
   23 CONTINUE
C
C
C          CONDENSE VRTX INTO SINGLE VERTEX
C          RELABEL VERTICES ACCORDING TO PRESENT CUT
C
      DO 24 J=1,N
      NODE(J)=0
   24 CONTINUE
C
C
      DO 26 I=1,COUNT2
C
      DO 25 J=1,N
      IF (J.NE.NOTVTX(ROW,I)) GO TO 25
      NODE(J)=1
      GO TO 26
   25 CONTINUE
C
   26 CONTINUE
C
      SOURCE=NOTVTX(ROW,1)
      SINK=NOTVTX(ROW,CCUNT2)
C
C          ERASE PRESENT VRTX AND NOTVTX ROWS
C
      DO 27 KK=1,N
      VRTX(ROW,KK)=0
      NOTVTX(ROW,KK)=0
   27 CONTINUE
C
      MARK=1
      FLAG=1
      ROW=ROW-1
      CALL MATRXC
      GO TO 6
C
C          COMPUTATION TERMINATES
   28 WRITE (6,31)
      WRITE (6,32)
      CALL PRTMAT
      STOP
C
   29 FORMAT (3I6)
   30 FORMAT (3512)
   31 FORMAT ('0',T14,'BRANCH-CONNECTIVITY MATRIX')
   32 FORMAT (T14,'--------------------------')
      END
C
C
C . . . . . SUBROUTINE MINCUT . . . . . .
C
C    THIS SUBROUTINE COMPUTES FOR THE MAXIMUM CONNECTIVITY
C BETWEEN TWO SELECTED VERTEX-PAIR USING THE MODIFIED FORD
C AND FULKERSON LABELING ALGORITHM.
C
```

```
      SUBROUTINE MINCUT
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(35,35), FLOMAT(35,35), CMAT(35,35)
      DIMENSION VRTX(35,35), NOTVTX(35,35), NODE(35)
      DIMENSION JSTORE(20), ISTORE(20)
      DIMENSION NLIST(35)
      COMMON /ARR1/ KMAT,FLOMAT,CMAT,VRTX,NOTVTX
      COMMON /CH1/ NODE,NLIST
      COMMON /PT1/ N,SOURCE,SINK,ROW,MARK,FLAG,BRCUT
      COMMON /PT2/ COUNT1,COUNT2,NCOUNT
C
C           INITIALIZE FLOW MATRIX TO ZERO
C
      DO 2 I=1,N
C
      DO 1 J=1,N
      FLOMAT(I,J)=0
    1 CONTINUE
C
    2 CONTINUE
C
C
C           ERASE VERTEX LABELS
C
    3 DO 4 J=1,N
      NODE(J)=0
    4 CONTINUE
C
C
      I=SOURCE
      M=0
      Z=0
      BRCUT=0
      NODE(I)=SOURCE
C
C           SEARCH FOR AUGMENTING PATH
C
    5 DO 6 J=1,N
C
C           TEST FOR CONNECTING BRANCH BETWEEN VERTEX-PAIR
      IF (CMAT(I,J).EQ.0) GO TO 6
      IF (FLOMAT(I,J).EQ.CMAT(I,J)) GO TO 6
C
C           IS NODE(J) LABELED?
      IF (NODE(J).NE.0) GO TO 6
      NODE(J)=I
C
C           IS THIS THE SINK VERTEX?
      IF (J.EQ.SINK) GO TO 9
      M=M+1
      JSTORE(M)=J
    6 CONTINUE
C
      IF (Z.NE.0) GO TO 8
      IF (M.EQ.0) GO TO 10
C
C           TRANSFER CONTENTS OF JSTORE TO ISTORE
C
      DO 7 S=1,M
      ISTORE(S)=JSTORE(S)
    7 CONTINUE
C
      Z=M
      M=0
    8 I=ISTORE(Z)
      Z=Z-1
      GO TO 5
C
C           WE REACH HERE IF AN AUGMENTING PATH IS FOUND
C             ADJUST FLOW ON ALL BRANCHES ALONG THIS PATH.
    9 I=NODE(J)
      FLOMAT(I,J)=FLOMAT(I,J)+1
```

```
      J=I
      IF (J.EQ.SOURCE) GO TO 3
      GO TO 9
C
C          THIS STEP IS REACHED IF LABELING AND FLOW AUGMENT-
C            ATION IS COMPLETED.  SEARCH FOR BRANCH CUTS.
C
   10 DO 13 I=1,N
      IF (NODE(I).EQ.0) GO TO 13
C
      DO 12 J=1,N
      ADD=1
      IF (CMAT(I,J).EQ.0) GO TO 12
      IF (NODE(J).NE.0) GO TO 12
C
C          IF FLOW OCCURS IN BOTH DIRECTIONS, NET FLOW IS ZERO
      IF (FLOMAT(I,J).EQ.FLOMAT(J,I)) GO TO 12
      IF (FLOMAT(I,J).EQ.1) GO TO 11
      ADD=FLOMAT(I,J)
   11 BRCUT=BRCUT+ADD
   12 CONTINUE
C
   13 CONTINUE
C
      RETURN
      END
C
C
C . . . . . SUBROUTINE MATRXT . . . . . .
C
C     THIS SUBROUTINE ENTERS COMPUTED CUT VALUE INTO A TERMI-
C NAL CAPACITY MATRIX TERMAT AND FURTHER UPDATES OTHER EN-
C TRIES BASED ON MAXIMUM FLOW ALONG A GIVEN PATH.  MATRIX
C TERMAT BECOMES THE OUTPUT MULTITERMINAL CONNECTIVITY MAT-
C RIX AFTER N-1 COMPUTATIONS.
C
      SUBROUTINE MATRXT
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(35,35), FLOMAT(35,35), CMAT(35,35)
      DIMENSION TERMAT(35,35)
      DIMENSION VRTX(35,35), NOTVTX(35,35), NODE(35)
      DIMENSION NX(35), NOTNX(35)
      DIMENSION NLIST(35)
      COMMON /ARR1/ KMAT,FLOMAT,CMAT,VRTX,NOTVTX
      COMMON /ARR2/ TERMAT
      COMMON /CH1/ NODE,NLIST
      COMMON /CH2/ NX,NOTNX
      COMMON /PT1/ N,SOURCE,SINK,ROW,MARK,FLAG,BRCUT
      COMMON /PT2/ COUNT1,COUNT2,NCOUNT
C
C          ENTER CUT VALUE
C
      DO 17 I=1,COUNT1
C
      DO 16 J=1,COUNT2
      KUT1=NX(I)
      KUT2=NOTNX(J)
      TERMAT(KUT1,KUT2)=BRCUT
      TERMAT(KUT2,KUT1)=BRCUT
C
C          COMPARE CUT VALUE WITH OTHER ENTRIES
C
      DO 15 LL=1,N
      IF (COUNT1.EQ.1) GO TO 7
      IF (COUNT2.EQ.1) GO TO 9
      IF (NODE(LL).NE.0) GO TO 15
      IF (FLAG.EQ.0) GO TO 8
C
C          CONDENSED VRTX IS ON THE SOURCE SIDE OF THE CUT
C
      DO 1 KK=1,NCOUNT
      IF (NLIST(KK).EQ.LL) GO TO 2
```

```
    1 CONTINUE
C
      GO TO 3
    2 X=NLIST(1)
      IF (NODE(X).EQ.0) GO TO 8
      GO TO 15
C
C          CONDENSED NOTVTX IS ON THE SOURCE SIDE OF THE CUT
    3 IKEEP=ROW
    4 IF (IKEEP.EQ.0) GO TO 8
C
      DO 5 KK=1,N
      IF (NOTVTX(IKEEP,KK).EQ.LL) GO TO 6
    5 CONTINUE
C
      IKEEP=IKEEP-1
      GO TO 4
    6 XBAR=NOTVTX(IKEEP,1)
      IF (NODE(XBAR).EQ.0) GO TO 8
      GO TO 15
    7 IF (NODE(LL).NE.0) GO TO 14
    8 IF (TERMAT(KUT1,LL).LE.TERMAT(KUT1,KUT2)) GO TO 15
      TERMAT(KUT1,LL)=BRCUT
      TERMAT(LL,KUT1)=BRCUT
      GO TO 15
    9 IF (NODE(LL).NE.0) GO TO 13
C
C          DISCARD THIS VERTEX IF COMPONENT OF SOME CONDENSED
C            VERTICES
C
      DO 10 JJ=1,NCOUNT
      IF (NLIST(JJ).EQ.LL) GO TO 13
   10 CONTINUE
C
      IKEEP=ROW
   11 IF (IKEEP.EQ.0) GO TO 15
C
      DO 12 JJ=1,N
      IF (NOTVTX(IKEEP,JJ).EQ.LL) GO TO 13
   12 CONTINUE
C
      IKEEP=IKEEP-1
      IF (IKEEP.EQ.0) GO TO 15
      GO TO 11
   13 IF (TERMAT(KUT2,LL).LE.TERMAT(KUT1,KUT2)) GO TO 15
      TERMAT(KUT2,LL)=BRCUT
      TERMAT(LL,KUT2)=BRCUT
      GO TO 15
   14 IF (COUNT2.EQ.1) GO TO 13
   15 CONTINUE
C
   16 CONTINUE
C
   17 CONTINUE
C
      RETURN
      END
C
C
C . . . . . SUBROUTINE MATRXC . . . . . .
C
C     THIS SUBROUTINE CONDENSES A GROUP OF VERTICES ON ONE
C SIDE OF A CUT INTO A GENERALIZED SINGLE VERTEX AND OB-
C TAINS A CAPACITY MATRIX CMAT REPRESENTING THE NEW GRAPH
C FORMED.
C
      SUBROUTINE MATRXC
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(35,35), FLOMAT(35,35), CMAT(35,35)
      DIMENSION VRTX(35,35), NOTVTX(35,35), NODE(35)
      DIMENSION NLIST(35)
      COMMON /ARR1/ KMAT,FLOMAT,CMAT,VRTX,NOTVTX
```

```
      COMMON /CH1/ NODE,NLIST
      COMMON /PT1/ N,SOURCE,SINK,ROW,MARK,FLAG,BRCUT
      COMMON /PT2/ COUNT1,COUNT2,NCOUNT
C
C          INITIALIZE CAPACITY MATRIX TO ZERO
C
      DO 2 I=1,N
C
      DO 1 J=1,N
      CMAT(I,J)=0
    1 CONTINUE
C
    2 CONTINUE
C
C
C          FILL UP CAPACITY MATRIX
C
      DO 13 I=1,N
      IF (NODE(I).EQ.0) GO TO 13
      IF (NCOUNT.EQ.0) GO TO 4
C
      DO 3 LL=1,NCOUNT
      IF (I.EQ.NLIST(LL)) GO TO 13
    3 CONTINUE
C
    4 MFLAG=0
      JFLAG=0
C
      DO 12 J=1,N
      IF (KMAT(I,J).EQ.0) GO TO 12
      IF (NODE(J).EQ.0) GO TO 5
      CMAT(I,J)=KMAT(I,J)
      GO TO 12
C
    5 INDEX=ROW
      IF (MARK.EQ.1) GO TO 6
      IF (FLAG.NE.0) GO TO 6
      GO TO 11
C
    6 DO 7 MM=1,NCOUNT
      IF (J.EQ.NLIST(MM)) GO TO 8
    7 CONTINUE
C
      GO TO 9
C
C          PROCESSED VERTICES (NLIST) TO BE CONDENSED
    8 MFLAG=MFLAG+1
      JTEMP=NLIST(1)
      CMAT(I,JTEMP)=MFLAG
      CMAT(JTEMP,I)=MFLAG
      GO TO 12
C
C          UNLABELED VERTICES TO BE CONDENSED
C
    9 DO 10 KK=1,N
      IF (J.EQ.NOTVTX(INDEX,KK)) GO TO 11
   10 CONTINUE
C
      INDEX=INDEX-1
      IF (INDEX.EQ.0) GO TO 12
      GO TO 9
   11 JFLAG=JFLAG+1
      JTEMP=NOTVTX(INDEX,1)
      CMAT(I,JTEMP)=JFLAG
      CMAT(JTEMP,I)=JFLAG
   12 CONTINUE
C
   13 CONTINUE
C
      RETURN
      END
C
```

```
C . . . . . SUBROUTINE PRTMAT . . . . . .
C
C     THIS SUBROUTINE PRINTS TERMAT IN PROPER FORM AND IN
C FORMAT 3513.
C
C   IF ARRAY LENGTH IS .LE. 35 A SINGLE LOOP SECTION PRINTS
C THE MATRIX, OTHERWISE IT PRINTS THE FIRST 35 COLUMNS FOL-
C LOWED BY THE NEXT 35 COLUMNS AND SO ON IN MULTIPLES OF 35
C USING MULTIPLE LOOPS. THE REMAINDER COLUMNS ARE PRINTED
C IMMEDIATELY BELOW THE LAST 35-COLUMN GROUP.
C
      SUBROUTINE PRTMAT
      IMPLICIT INTEGER(A-Z)
      DIMENSION TERMAT(35,35)
      COMMON /ARR2/ TERMAT
      COMMON /PTI/ N,SOURCE,SINK,ROW,MARK,FLAG,BRCUT
C
C          HOW MANY MULTIPLES OF 35 IS N?
      MULPLE=N/35
      IF (MULPLE.GE.1) GO TO 2
C
C          THIS SECTION PRINTS ARRAY WITH LENGTH .LT. 35
C
      DO 1 I=1,N
      WRITE (6,8) I,(TERMAT(I,J),J=1,N)
    1 CONTINUE
C
      GO TO 7
C
C          THIS SECTION PRINTS ARRAY WITH LENGTH .GE. 35
C            IN GROUPS OF 35
    2 M=1
      K=1
    3 MPLS35=M+35
C
      DO 4 I=1,N
      WRITE (6,8) I,(TERMAT(I,J),J=M,MPLS35)
    4 CONTINUE
C
      M=M+35
      IF (K.EQ.MULPLE) GO TO 5
      WRITE (6,9) K
      WRITE (6,10)
      K=K+1
      GO TO 3
C
C          THIS SECTION PRINTS REMAINDER COLUMNS
    5 REMDR=N-MULPLE*35
      IF (REMDR.LE.0) GO TO 7
      WRITE (6,11)
      WRITE (6,12)
C
      DO 6 L=1,N
      WRITE (6,8) L,(TERMAT(L,J),J=M,N)
    6 CONTINUE
C
    7 RETURN
C
    8 FORMAT ('0', T8, I2, '/', 1X, 35I3)
    9 FORMAT (//T14,'CONTINUATION', I3, 2X, 'OF TERMAT')
   10 FORMAT (T14,'-------------------------')
   11 FORMAT (//T14,'LAST SECTION OF TERMAT')
   12 FORMAT (T14,'-----------------------')
      END
```

B-4. PROGRAM MAXCON-2A

```
C    THIS PROGRAM COMPUTES FOR THE MAXIMUM BRANCH-CONNECT-
C IVITY BETWEEN ANY TWO GIVEN VERTICES OF AN UNDIRECTED
C NETWORK. THE NETWORK GRAPH IS REPRESENTED BY ITS CONNECT-
C ION MATRIX FOR COMPUTATION. THE NETWORK CAN HAVE PARALLEL
C BRANCHES BETWEEN VERTICES.
C
C    THE PROGRAM STARTS BY READING-IN THE NUMBER OF VERTICES,
C THE SOURCE AND SINK VERTICES, AND THE CONNECTION MATRIX.
C THE CONNECTIVITY OF A CUT(X,XBAR) IS DETERMINED USING THE
C MODIFIED FORD AND FULKERSON LABELING ALGORITHM.
C
C    THE OUTPUT CONSIST OF THE VALUE OF THE CONNECTIVITY AND
C THE VERTEX PAIRS JOINING EACH BRANCH CUT.
C
C    THIS PROGRAM MAYBE PREFERRED OVER MAXCON-2 WHEN ONLY
C SEVERAL SOURCE-SINK PAIRS ARE TO BE INVESTIGATED.
C
C    THE PROGRAM DIMENSION AND READ STATEMENTS MUST BE
C ADJUSTED FOR LARGER NETWORKS.
C
C
      IMPLICIT INTEGER(A-Z)
C     DIMENSION KMAT(60,60), FLOMAT(60,60), NODE(60), NX(20)
      DIMENSION NOTNX(20), JSTORE(20), ISTORE(20)
C
      READ (5,17) N
      READ (5,18) ((KMAT(I,J),J=1,60),I=1,N)
      WRITE (6,20)
    1 READ (5,19) SOURCE,SINK
      IF (SOURCE.EQ.0) GO TO 16
C
C          INITIALIZE FLOW MATRIX TO ZERO
C
      DO 3 I=1,N
C
      DO 2 J=1,N
      FLOMAT(I,J)=0
    2 CONTINUE
C
    3 CONTINUE
C
C
C          ERASE VERTEX LABELS
C
    4 DO 5 J=1,N
      NODE(J)=0
    5 CONTINUE
C
C
      I=SOURCE
      L=0
      M=0
      Z=0
      BRCUT=0
      NODE(I)=SOURCE
C
C          SEARCH FOR AUGMENTING PATH
C
    6 DO 7 J=1,N
C
C          TEST FOR CONNECTING BRANCH BETWEEN VERTEX-PAIR
      IF (KMAT(I,J).EQ.0) GO TO 7
      IF (FLOMAT(I,J).EQ.KMAT(I,J)) GO TO 7
C
C          IS NODE(J) LABELED?
      IF (NODE(J).NE.0) GO TO 7
      NODE(J)=I
C
```

```
C          IS THIS THE SINK VERTEX?
      IF (J.EQ.SINK) GO TO 10
      M=M+1
      JSTORE(M)=J
    7 CONTINUE
C
      IF (Z.NE.0) GO TO 9
      IF (M.EQ.0) GO TO 11
C
C          TRANSFER CONTENTS OF JSTORE TO ISTORE
C
      DO 8 S=1,M
      ISTORE(S)=JSTORE(S)
    8 CONTINUE
C
      Z=M
      M=0
    9 I=ISTORE(Z)
      Z=Z-1
      GO TO 6
C
C          WE REACH HERE IF AN AUGMENTING PATH IS FOUND
C            ADJUST FLOW ON ALL BRANCHES ALONG THIS PATH.
   10 I=NODE(J)
      FLOMAT(I,J)=FLOMAT(I,J)+1
      J=I
      IF (J.EQ.SOURCE) GO TO 4
      GO TO 10
C
C          THIS STEP IS REACHED IF LABELING AND FLOW AUGMENT-
C            ATION IS COMPLETED.  SEARCH FOR BRANCH CUTS.
C
   11 DO 14 I=1,N
      IF (NODE(I).EQ.0) GO TO 14
C
      DO 13 J=1,N
      ADD=1
      IF (KMAT(I,J).EQ.0) GO TO 13
      IF (NODE(J).NE.0) GO TO 13
C
C          IF FLOW OCCURS IN BOTH DIRECTIONS, NET FLOW IS ZERO
      IF (FLOMAT(I,J).EQ.FLOMAT(J,I)) GO TO 13
      IF (FLOMAT(I,J).EQ.1) GO TO 12
      ADD=FLOMAT(I,J)
   12 BRCUT=BRCUT+ADD
      L=L+1
      NX(L)=I
      NOTNX(L)=J
   13 CONTINUE
C
   14 CONTINUE
C
C
C          COMPUTATION TERMINATES SINCE NO MORE AUGMENTING
C            PATH IS FOUND.
      IF (BRCUT.NE.0) GO TO 15
      WRITE (6,21) SOURCE,SINK,BRCUT
      GO TO 1
   15 WRITE (6,22) SOURCE,SINK,BRCUT,(NX(J),NOTNX(J),J=1,L)
      GO TO 1
   16 STOP
C
   17 FORMAT (I6)
   18 FORMAT (3512,/25I2)
   19 FORMAT (216)
   20 FORMAT ('0', 10X, 'SOURCE', 10X, 'SINK', 10X, 'BR-
     1     CONNECTIVITY', 10X, 'BR-CUTSET')
   21 FORMAT ('0', 12X, I2, 13X, I2, 17X, I2)
   22 FORMAT ('0', 12X, I2, 13X, I2, 17X, I2,
     1     (T67, I3, 1X, '--', I3/))
      END
```

B-5. PROGRAM MAXCON-3

```
C    THIS PROGRAM COMPUTES FOR THE MULTITERMINAL  VERTEX-
C  CONNECTIVITY OF A DIRECTED OR MIXED NETWORK. THE NETWORK
C  IS REPRESENTED BY ITS CONNECTION MATRIX FOR COMPUTATION.
C  THE NETWORK MUST HAVE ONLY SINGLE BRANCH BETWEEN VERTICES
C
C    THE PROGRAM PROCEEDS BY READING-IN THE INPUT DATA CON-
C  SISTING OF THE TOTAL NUMBER OF VERTICES AND THE CONNECT-
C  ION MATRIX. IT THEN DETERMINES THE INITIAL S'-T CUT WITH
C  THE FIRST AND NTH VERTICES AS SOURCE AND SINK RESP.
C  HAVING FOUND AN S'-T CUT, IT THEN FINDS OTHER S-T CUTS
C  KEEPING THE SINK VERTEX CONSTANT AND VARYING ONLY THE
C  SOURCE VERTEX FROM 2 TO N. AFTER COMPLETING THIS LOOP,
C  THE SINK VERTEX IS SET TO N-1 AND THEN FINDS AN S'-T CUT
C  BETWEEN VERTICES 1 AND N-1. THE SOURCE IS AGAIN VARIED
C  FROM 2 TO N-1 FOR THE OTHER S-T CUTS.
C
C    THE PROCESS IS REPEATED UNTIL N(N-1) CUTS ARE DETER-
C  MINED. EACH REPETITION INVOLVES SUBROUTINE FLOVAR.
C
C    THE OUTPUT IS A PRINTOUT OF THE NXN VERTEX-CONNECTIVITY
C  MATRIX.
C
C  . . . . . . . . IDENTIFIER NOMENCLATURE . . . . . . . . .
C
C   ARRAYS
C      KMAT--- INPUT CONNECTION MATRIX WITH ENTRIES 0, 1, -1
C      DMAT--- S'-T FLOW MATRIX REPRESENTING AN S'-T FLOW
C         PATTERN. THIS MATRIX IS UPDATED BEFORE A NEW SET
C         OF S AND S' VERTICES ARE DESIGNATED.
C      FLOMAT- S-T FLOW MATRIX REPRESENTING S-T FLOW PAT-
C         TERN RESULTING FROM AUGMENTATION AND HOMING STEPS.
C         NONZERO ENTRIES OF THIS MATRIX UPDATE DMAT.
C      TERMAT- MATRIX DENOTING THE MAXIMUM BRANCH-CONNECT-
C         IVITY BETWEEN ANY VERTEX-PAIR.
C
C   VECTOR
C      NODE--- LIST OF VERTICES OF THE NETWORK GRAPH.
C
C  . . . . . . . . ADDITIONAL INFORMATION . . . . . . . . .
C
C    THIS PROGRAM IS CODED IN FORTRAN LANGUAGE. NETWORK OF
C  ANY SIZE CAN BE HANDLED BY THIS PROGRAM WITHIN THE LIMIT-
C  ATIONS OF THE COMPUTER STORAGE. THE DIMENSION, READ, AND
C  WRITE STATEMENTS MUST HOWEVER BE ADJUSTED ACCORDINGLY.
C
C
C  . . . . . . . . . . . . MAIN PROGRAM . . . . . . . . . . .
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(25,25), FLOMAT(25,25), DMAT(25,25)
      DIMENSION TERMAT(25,25)
      DIMENSION NODE(25)
      COMMON /ARR1/ KMAT,FLOMAT,DMAT
      COMMON /ARR2/ TERMAT
      COMMON /CH/ NODE
      COMMON /PT/ SOURCE,SPRIME,SINK,AMARK,VTXCUT,N,G,FLAG
C
      READ (5,11) N
      READ (5,12) ((KMAT(I,J),J=1,25),I=1,N)
C
C         INITIALIZE TERMINAL MATRIX
C
      DO 2 X=1,N
C
      DO 1 Y=1,25
      TERMAT(X,Y)=0
    1 CONTINUE
C
```

```
    2 CONTINUE
C
C
C           COMPUTE FOR THE N(N-1) VERTEX CUTS
C
      DO 10 K=1,N
      J=N+1-K
C
C           SET TO ZERO THE FLOW MATRICES
C
      DO 4 L=1,N
C
      DO 3 M=1,N
      DMAT(L,M)=0
      FLOMAT(L,M)=0
    3 CONTINUE
C
    4 CONTINUE
C
      SPRIME=0
      FLAG=0
C
      DO 9 I=1,N
      IF (I.EQ.J) GO TO 9
C
C           IS THERE A DIRECT CONNECTION BETWEEN SOURCE AND
C             SINK?
      IF (KMAT(I,J).GT.0) GO TO 5
      SOURCE=I
      SINK=J
      VTXCUT=0
      CALL FLOVAR
C
C           ENTER CUT VALUE INTO THE TERMINAL MATRIX
      TERMAT(I,J)=VTXCUT
      SPRIME=I
      GO TO 6
    5 TERMAT(I,J)=0
      GO TO 9
C
C           TRANSFER FLOMAT ENTRIES INTO DMAT
C
    6 DO 8 L=1,N
C
      DO 7 M=1,N
      IF (FLOMAT(L,M).EQ.0) GO TO 7
      DMAT(L,M)=FLOMAT(L,M)
C
C           RESET FLOMAT TO ZERO
      FLOMAT(L,M)=0
    7 CONTINUE
C
    8 CONTINUE
C
      FLAG=1
    9 CONTINUE
C
   10 CONTINUE
C
C
C           COMPUTATION COMPLETED
      WRITE (6,13)
      WRITE (6,14)
      CALL PRTMAT
      STOP
C
   11 FORMAT (I6)
   12 FORMAT (25I3)
   13 FORMAT ('0',T14,'VERTEX-CONNECTIVITY MATRIX')
   14 FORMAT (T14,'--------------------------')
      END
C
```

```
C . . . . . SUBROUTINE FLOVAR . . . . . .
C
C   THIS SUBROUTINE DETERMINES AN S-T CUT ONCE AN INITIAL
C S'-T CUT IS FOUND. IT MAXIMIZES AN S-T FLOW USING THE
C EXISTING S'-T FLOW PATTERN.
C
      SUBROUTINE FLOVAR
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(25,25), FLOMAT(25,25), DMAT(25,25)
      DIMENSION NODE(25)
      COMMON /ARR1/ KMAT,FLOMAT,DMAT
      COMMON /CH/ NODE
      COMMON /PT/ SOURCE,SPRIME,SINK,AMARK,VTXCUT,N,G,FLAG
C
C          ERASE ALL VERTEX LABELS
C
    1 DO 2 J=1,N
      NODE(J)=0
    2 CONTINUE
C
C
C          SEARCH FOR AUGMENTATION PATH
      CALL VTXCON
      IF (AMARK) 3,24,15
C
C          AN S-G AUGMENTATION PATH IS FOUND
C          SEARCH FOR HOMING PATH
    3 Q=G
C
    4 DO 5 J=1,N
      IF (DMAT(Q,J).EQ.1) GO TO 6
    5 CONTINUE
C
      GO TO 1
    6 NODE(J)=Q
      Q=J
      IF (Q.EQ.G) GO TO 19
      IF (J.EQ.SINK) GO TO 7
      GO TO 4
C
C          A HOMING PATH IS FOUND
C          SEARCH FOR TRUNCATION PATH
    7 Z=G
      IF (G.EQ.SPRIME) GO TO 12
C
    8 DO 9 I=1,N
      IF (DMAT(I,Z).EQ.1) GO TO 10
    9 CONTINUE
C
      IF (Z.EQ.G) GO TO 12
      GO TO 11
   10 NODE(I)=Z
      Z=I
      IF (Z.EQ.G) GO TO 21
      IF (Z.EQ.SPRIME) GO TO 11
      GO TO 8
C
C          A TRUNCATION PATH IS FOUND
C          TRUNCATE SPRIME-G PATH
   11 I=NODE(Z)
      DMAT(Z,I)=0
      Z=I
      IF (Z.EQ.G) GO TO 12
      GO TO 11
C
C          TRACE AND ERASE G-T PATH
   12 Q=SINK
   13 J=NODE(Q)
      FLOMAT(J,Q)=1
      DMAT(J,Q)=0
      Q=J
      IF (J.EQ.G) GO TO 14
```

```
      GO TO 13
C
C          AUGMENT S-G PATH
   14 IF (G.EQ.SOURCE) GO TO 23
      J=G
      GO TO 16
C
C          AUGMENT S-T PATH
   15 J=SINK
   16 I=NODE(J)
      IF (I.LT.0) GO TO 17
C
C          VERTEX IS WEAKLY LABELED. INCREMENT FWD FLOW BY 1
      FLOMAT(I,J)=1
      GO TO 18
C
C          VERTEX IS STRONGLY LABELED. DECREMENT BACKFLOW BY 1
   17 I=-I
      FLOMAT(J,I)=0
   18 J=I
C
C          IS SOURCE VERTEX REACHED?
      IF (J.EQ.SOURCE) GO TO 23
      GO TO 16
C
C          A LOOP OF FLOW IS FOUND BY HOMING ROUTINE
   19 Q=G
   20 J=NODE(Q)
      DMAT(J,Q)=0
      Q=J
      IF (Q.EQ.G) GO TO 1
      GO TO 20
C
C          A LOOP OF FLOW IS FOUND BY TRUNCATION ROUTINE
   21 Z=G
   22 I=NODE(Z)
      DMAT(Z,I)=0
      Z=I
      IF (Z.EQ.G) GO TO 1
      GO TO 22
   23 VTXCUT=VTXCUT+1
      GO TO 1
C
C          NO MORE AUGMENTATION PATH IS FOUND. FLOW PATTERN
C            IS MAX
   24 RETURN
      END
C
C
C . . . . . .  SUBROUTINE VTXCON  . . . . . .
C
C   THIS SUBROUTINE LOCATES AN S-G OR S-T AUGMENTATION PATH
C USING FRISCH'S VERTEX-PAIR CONNECTIVITY ALGORITHM.
C
C   IT IS INVOKED BY SUBROUTINE FLOVAR WHICH SPECIFIES THE
C SOURCE AND SINK VERTICES. THIS ROUTINE LABELS THE VERTI-
C CES WHILE LOCATING A PATH. INFORMATION WHETHER THERE IS
C AN AUGMENTING PATH OR NOT IS RETURNED TO ROUTINE FLOVAR.
C
      SUBROUTINE VTXCON
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(25,25), FLOMAT(25,25), DMAT(25,25)
      DIMENSION NODE(25), JSTORE(20), ISTORE(20)
      COMMON /AREA/ KMAT,FLOMAT,DMAT
      COMMON /CH/ NODE
      COMMON /PT/ SOURCE,SPRIME,SINK,AMARK,VTXCUT,N,G,FLAG
C
C          INITIALIZE VARIABLES
      I=SOURCE
      L=0
      M=0
      Z=0
```

```
      AMARK=0
      NODE(I)=SOURCE
C
C          SEARCH FOR AUGMENTING PATH
C
    1 DO 6 J=1,N
C
C          TEST FOR CONNECTING PATH BETWEEN VERTEX-PAIR
      IF (FLOMAT(J,I).EQ.0) GO TO 3
      IF (NODE(I).LT.0) GO TO 2
C
C          A BACKWARD BRANCH IS ENCOUNTERED.
      IF (J.EQ.SOURCE) GO TO 7
      IF (NODE(J).LT.0) GO TO 6
C
C          VERTEX(J) IS EITHER UNLABELED OR WEAKLY LABELED.
C            STRONGLY LABEL THIS VERTEX.
      NODE(J)=-I
      M=M+1
      JSTORE(M)=J
      GO TO 7
    2 IF (J.EQ.SOURCE) GO TO 6
      NODE(J)=-I
      GO TO 5
C
    3 IF (KMAT(I,J).NE.1) GO TO 6
C
C          A FORWARD FLOW IS ENCOUNTERED. TEST FOR ZERO FLOW.
      IF (FLAG.EQ.0) GO TO 4
      IF (DMAT(I,J).NE.0) GO TO 10
    4 IF (FLOMAT(I,J).NE.0) GO TO 6
      IF (NODE(J).NE.0) GO TO 6
      NODE(J)=I
C
C          IS SINK VERTEX REACHED?
      IF (J.EQ.SINK) GO TO 11
    5 M=M+1
      JSTORE(M)=J
    6 CONTINUE
C
    7 IF (Z.NE.0) GO TO 9
      IF (M.EQ.0) GO TO 12
C
C          TRANSFER CONTENTS OF JSTORE INTO ISTORE
C
      DO 8 S=1,M
      ISTORE(S)=JSTORE(S)
    8 CONTINUE
C
      Z=M
      M=0
    9 I=ISTORE(Z)
      Z=Z-1
      GO TO 1
C
C          AN S-G AUGMENTATION PATH IS FOUND
   10 G=I
      AMARK=-1
      GO TO 13
C
C     AN S-T AUGMENTATION PATH IS FOUND
   11 AMARK=1
      GO TO 13
C
C          NO AUGMENTATION PATH FOUND
   12 AMARK=0
   13 RETURN
      END
C
C
C
C
```

```
C . . . . . SUBROUTINE PRTMAT . . . . . .
C
C     THIS SUBROUTINE PRINTS TERMAT IN PROPER FORM AND IN
C FORMAT 3513.
C
C   IF ARRAY LENGTH IS .LE. 35 A SINGLE LOOP SECTION PRINTS
C THE MATRIX, OTHERWISE IT PRINTS THE FIRST 35 COLUMNS FOL-
C LOWED BY THE NEXT 35 COLUMNS AND SO ON IN MULTIPLES OF 35
C USING MULTIPLE LOOPS. THE REMAINDER COLUMNS ARE PRINTED
C IMMEDIATELY BELOW THE LAST 35-COLUMN GROUP.
C
      SUBROUTINE PRTMAT
      IMPLICIT INTEGER(A-Z)
      DIMENSION TERMAT(25,25)
      COMMON /ARR2/ TERMAT
      COMMON /PT/ SOURCE,SPRIME,SINK,AMARK,VTXCUT,N,G,FLAG
C
C          HOW MANY MULTIPLES OF 35 IS N?
      MULPLE=N/35
      IF (MULPLE.GE.1) GO TO 2
C
C          THIS SECTION PRINTS ARRAY WITH LENGTH .LT. 35
C
      DO 1 I=1,N
      WRITE (6,8) I,(TERMAT(I,J),J=1,N)
    1 CONTINUE
C
      GO TO 7
C
C          THIS SECTION PRINTS ARRAY WITH LENGTH .GE. 35
C            IN GROUPS OF 35
    2 M=1
      K=1
    3 MPLS35=M+35
C
      DO 4 I=1,N
      WRITE (6,8) I,(TERMAT(I,J),J=M,MPLS35)
    4 CONTINUE
C
      M=M+35
      IF (K.EQ.MULPLE) GO TO 5
      WRITE (6,9) K
      WRITE (6,10)
      K=K+1
      GO TO 3
C
C          THIS SECTION PRINTS REMAINDER COLUMNS
    5 REMDR=N-MULPLE*35
      IF (REMDR.LE.0) GO TO 7
      WRITE (6,11)
      WRITE (6,12)
C
      DO 6 L=1,N
      WRITE (6,8) L,(TERMAT(L,J),J=M,N)
    6 CONTINUE
C
    7 RETURN
C
    8 FORMAT ('0', T8, I2, '/', 1X, 35I3)
    9 FORMAT (//T14,'CONTINUATION', I3, 2X, 'OF TERMAT')
   10 FORMAT (T14,'-------------------------')
   11 FORMAT (//T14,'LAST SECTION OF TERMAT')
   12 FORMAT (T14,'-----------------------')
      END
```

B-6. PROGRAM MAXCON-3A

```
C    THIS PROGRAM COMPUTES FOR THE VERTEX-CONNECTIVITY OF A
C  DIRECTED, AN UNDIRECTED, OR MIXED NETWORK USING FRISCH'S
C  VERTEX-PAIR CONNECTIVITY ALGORITHM.
C
C    THE PROGRAM STARTS BY READING-IN THE INPUT DATA CONSIS-
C  TING OF THE NUMBER OF VERTICES, THE CONNECTION MATRIX,
C  AND THE SOURCE AND SINK VERTICES AS CHOSEN. THENCE IT
C  FINDS AN AUGMENTING PATH USING THE MODIFIED LABELING AL-
C  GORITHM AND INCREASES/DECREASES THE FLOW ONCE SUCH PATH
C  IS FOUND. THE PROCESS IS REPEATED UNTIL NO MORE AUGMENT-
C  ING PATH IS FOUND. IT THEN SEARCHES FOR THE VERTEX CUTSET
C  BY FINDING A LABELED VERTEX AND AN UNLABELED ONE WHICH
C  ARE BOTH CONNECTED AND HAVE UNITY FLOW BETWEEN THEM.
C
C    THE OUTPUT IS A PRINTOUT OF THE SOURCE VERTEX, THE SINK
C  VERTEX, AND A LIST OF VERTICES COMPRISING THE CUTSET.
C
C    FOR NETWORKS LARGER THAN THE DECLARED DIMENSION, THE
C  DIMENSION AND READ STATEMENTS MUST BE ADJUSTED.
C
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(50,50), FLOMAT(50,50)
      DIMENSION NODE(50), JSTORE(20), ISTORE(20), VTXCUT(10)
      READ (5,24) N
      READ (5,25) ((KMAT(I,J),J=1,50),I=1,N)
      WRITE (6,27)
    1 READ (5,26) SOURCE,SINK
      IF (SOURCE.EQ.0) GO TO 23
C
CC         IS THERE A DIRECT CONNECTION BETWEEN SOURCE
C            AND SINK?
      IF (KMAT(SOURCE,SINK).GT.0) GO TO 22
C
CC         INITIALIZE FLOW MATRIX TO ZERO
C
      DO 3 I=1,N
C
      DO 2 J=1,N
      FLOMAT(I,J)=0
    2 CONTINUE
C
    3 CONTINUE
C
CC
CC         ERASE VERTEX LABELS
C
    4 DO 5 J=1,N
      NODE(J)=0
    5 CONTINUE
C
CC
C          INITIALIZE VARIABLES
      I=SOURCE
      L=0
      M=0
      Z=0
      NODE(I)=SOURCE
C
CC         SEARCH FOR AUGMENTING PATH
C
    6 DO 10 J=1,N
C
C          TEST FOR CONNECTING PATH BETWEEN VERTEX-PAIR
      IF (FLOMAT(J,I).EQ.0) GO TO 8
      IF (NODE(I).LT.0) GO TO 7
C
C          A BACKWARD BRANCH IS ENCOUNTERED.
```

```
      IF (J.EQ.SOURCE) GO TO 11
      IF (NODE(J).LT.0) GO TO 10
C
C          VERTEX(J) IS EITHER UNLABELED OR WEAKLY LABELED.
C            STRONGLY LABEL THIS VERTEX.
      NODE(J)=-I
      M=M+1
      JSTORE(M)=J
      GO TO 11
    7 IF (J.EQ.SOURCE) GO TO 10
      NODE(J)=-I
      GO TO 9
C
    8 IF (KMAT(I,J).NE.1) GO TO 10
C
C          A FORWARD BRANCH IS ENCOUNTERED. TEST FOR 0 FLOW.
      IF (FLOMAT(I,J).NE.0) GO TO 10.
      IF (NODE(J).NE.0) GO TO 10
      NODE(J)=I
C
C          IS SINK VERTEX REACHED?
      IF (J.EQ.SINK) GO TO 14
    9 M=M+1
      JSTORE(M)=J
   10 CONTINUE
C
   11 IF (Z.NE.0) GO TO 13
      IF (M.EQ.0) GO TO 17
C
C          TRANSFER CONTENTS OF JSTORE INTO ISTORE
C
      DO 12 S=1,M
      ISTORE(S)=JSTORE(S)
   12 CONTINUE
C
      Z=M
      M=0
   13 I=ISTORE(Z)
      Z=Z-1
      GO TO 6
C
C          WE REACH HERE IF AN AUGMENTING PATH IS FOUND.
C            ADJUST ALL FLOWS ALONG THIS PATH.
C          TEST LABEL ON VERTEX(J)
   14 I=NODE(J)
      IF (I.LT.0) GO TO 15
C
C          VERTEX IS WEAKLY LABELED. INCREMENT FWD FLOW BY 1
      FLOMAT(I,J)=1
      GO TO 16
C
C          VERTEX IS STRONGLY LABELED. DECREMENT BACKFLOW
   15 I=-I
      FLOMAT(J,I)=0
   16 J=I
C
C          IS SOURCE VERTEX REACHED?
      IF (J.EQ.SOURCE) GO TO 4
      GO TO 14
C
C
C          VERTEX LABELING AND FLOW AUGMENTATION IS COMPLTD.
C            THIS SECTION SEARCHES FOR THE VERTEX CUTS
C
   17 DO 20 I=1,N
      IF (NODE(I).EQ.0) GO TO 20
C
      DO 19 J=1,N
      IF (NODE(J).NE.0) GO TO 19
      IF (FLOMAT(I,J).EQ.0) GO TO 19
      IF (I.EQ.SOURCE) GO TO 18
      L=L+1
```

```
      VTXCUT(L)=I
      GO TO 20
   18 L=L+1
      VTXCUT(L)=J
   19 CONTINUE
C
   20 CONTINUE
C
C
C
C           COMPUTATION TERMINATES SINCE NO MORE AUGMENTING
C             PATH IS FOUND. PRINT CUTSET.
      IF (L.EQ.0) GO TO 21
      WRITE (6,28) SOURCE,SINK,(VTXCUT(J),J=1,L)
      GO TO 1
   21 WRITE (6,29) SOURCE,SINK
      GO TO 1
   22 WRITE (6,30) SOURCE,SINK
      GO TO 1
   23 STOP
C
   24 FORMAT (I6)
   25 FORMAT (25I3/25I3)
   26 FORMAT (2I6)
   27 FORMAT ('0', 10X, 'SOURCE', 10X, 'SINK', 10X,
     1    'VERTEX-CUTSET')
   28 FORMAT ('0', 12X, I2, 13X, I2, (T42, 4I3)/)
   29 FORMAT ('0',12X,I2,13X,I2,T41,'NO FEASIBLE FLOW')
   30 FORMAT ('0', 12X, I2, 13X, I2, T44, '0')
      END
```

B-7. PROGRAM MAXCON-4

```
C    THIS PROGRAM VERIFIES THE R-CONNECTIVITY OF AN UNDIR-
C ECTED NETWORK FOR A GIVEN UNIFORM REQUIREMENT R.
C
C    INPUT DATA ARE THE NUMBER OF VERTICES, THE REQUIREMENT
C R, AND THE NETWORK CONNECTION MATRIX.
C
C    THE PROGRAM READS-IN THE INPUT DATA THEN DETERMINES THE
C NUMBER OF VERTEX DISJOINT PATHS, STARTING WITH VERTEX 1
C AS SOURCE, TO EACH OF THE REMAINING VERTICES. IT EMPLOYS
C FRISCH'S ALGORITHM FOR VERTEX-PAIR CONNECTIVITY AND
C KLEITMAN'S THEOREMS FOR OBTAINING THE OVERALL CONNECTI-
C VITY.
C
C    IF A CONNECTIVITY LESS THAN THE REQUIREMENT IS ENCOUNT-
C ERED, THE PROGRAM CEASES FROM FURTHER VERIFICATION AND
C PRINTS OUT THE CONNECTIVITY VALUE SPECIFYING THE CORRESP-
C ONDING SOURCE VERTEX. OTHERWISE THE PROGRAM WILL INDICATE
C THAT THE NETWORK IS AT LEAST R-CONNECTED.
C
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(50,50), ILIST(10)
      COMMON /ARR/ KMAT
      COMMON /CHAIN/ ILIST
      COMMON /PT/ SOURCE,SINK,LL,Q,N,RTEMP
C
      READ (5,7) N,R
      READ (5,8) ((KMAT(I,J),J=1,50),I=1,N)
      L=0
      INDEX=0
      RTEMP=R
C
      DO 4 I=1,R
      IPLUS1=I+1
C
      DO 3 J=IPLUS1,N
C
C          LIST DOWN PREVIOUSLY DESIGNATED SINK VERTICES THAT
C            ARE CONNECTED TO VERTEX J.
      LL=0
      JLESS1=J-1
      IF (JLESS1.LT.IPLUS1) GO TO 2
C
      DO 1 L=IPLUS1,JLESS1
      IF (KMAT(L,J).EQ.0) GO TO 1
      LL=LL+1
      ILIST(LL)=L
    1 CONTINUE
C
      IF (LL.GE.RTEMP) GO TO 3
C
C          SEARCH FOR DISJOINT PATHS
    2 SOURCE=I
      SINK=J
      Q=0
      CALL VTXCON
      S=Q+LL+INDEX
      IF (S.GE.R) GO TO 3
      GO TO 5
C
C          ARE ALL (N-I) VERTICES PROCESSED?
    3 CONTINUE
C
      RTEMP=RTEMP-I
      INDEX=INDEX+1
    4 CONTINUE
C
C
```

```
C          VERIFICATION OF RTH VERTEX AS SOURCE COMPLETED.
      WRITE (6,9) R
      GO TO 6
C
C          CONNECTIVITY IS LESS THAN R. TERMINATE.
    5 WRITE (6,10) S
      WRITE (6,11) I
    6 STOP
C
    7 FORMAT (2I6)
    8 FORMAT (25I3/25I3)
    9 FORMAT ('0',T10,'NETWORK IS AT LEAST',I3,'-CONNECTED')
   10 FORMAT ('0',T10,'NETWORK IS ONLY',I3,'-CONNECTED WITH'
   11 FORMAT ('0', T13, 'VERTEX',I3,2X,'AS SOURCE')
      END
C
C
C . . . . . SUBROUTINE VTXCON . . . . . .
C
C   THIS SUBROUTINE DETERMINES THE NUMBER OF VERTEX DIS-
C JOINT PATHS BETWEEN TWO GIVEN VERTICES USING FRISCH'S AL-
C GORITHM. IT IS HOWEVER MODIFIED BY INCORPORATING KLEIT-
C MAN'S THEOREMS TO REDUCE COMPUTATIONAL TIME.
C
      SUBROUTINE VTXCON
      IMPLICIT INTEGER(A-Z)
      DIMENSION KMAT(50,50), FLOMAT(50,50)
      DIMENSION NODE(50), JSTORE(20), ISTORE(20), ILIST(10)
      COMMON /ARR/ KMAT
      COMMON /CHAIN/ ILIST
      COMMON /PT/ SOURCE,SINK,LL,Q,N,RTEMP
C
C          INITIALIZE FLOW MATRIX TO ZERO
C
      DO 2 I=SOURCE,N
C
      DO 1 J=SOURCE,N
      FLOMAT(I,J)=0
    1 CONTINUE
C
    2 CONTINUE
C
      MARK=0
C
C          ERASE VERTEX LABELS
C
    3 DO 4 J=SOURCE,N
      NODE(J)=0
    4 CONTINUE
C
C
C          INITIALIZE VARIABLES
      I=SOURCE
      L=0
      M=0
      Z=0
      NODE(I)=SOURCE
      SPLUS1=SOURCE+1
C
C          IS SOURCE CONNECTED TO SINK?
      IF (KMAT(SOURCE,SINK).EQ.0) GO TO 5
      KMAT(SOURCE,SINK)=0
      KMAT(SINK,SOURCE)=0
      Q=Q+1
      MARK=1
C
C          SEARCH FOR AUGMENTING PATH
C
    5 DO 11 J=SPLUS1,N
C
C          REMOVE PROCESSED VERTICES THAT ARE CONNECTED TO
C            SINK
```

```
      IF (LL.EQ.0) GO TO 7
C
      DO 6 K=1,LL
      IF (J.EQ.ILIST(K)) GO TO 11
    6 CONTINUE
C
C
C          TEST FOR CONNECTING PATH BETWEEN VERTEX-PAIR
    7 IF (FLOMAT(J,I).EQ.0) GO TO 9
      IF (NODE(I).LT.0) GO TO 8
C
C          A BACKWARD BRANCH IS ENCOUNTERED
      IF (NODE(J).LT.0) GO TO 11
      IF (J.EQ.SOURCE) GO TO 12
C
C          VERTEX(J) IS EITHER UNLABELED OR WEAKLY LABELED.
C            STRONGLY LABEL THIS VERTEX
      NODE(J)=-I
      M=M+1
      JSTORE(M)=J
      GO TO 12
    8 IF (J.EQ.SOURCE) GO TO 11
      NODE(J)=-I
      GO TO 10
    9 IF (KMAT(I,J).NE.1) GO TO 11
C
C          A FORWARD BRANCH IS ENCOUNTERED. TEST FOR ZERO FLOW
      IF (FLOMAT(I,J).NE.0) GO TO 11
      IF (NODE(J).NE.0) GO TO 11
      NODE(J)=I
C
C          IS SINK VERTEX REACHED?
      IF (J.EQ.SINK) GO TO 15
   10 M=M+1
      JSTORE(M)=J
   11 CONTINUE
C
   12 IF (Z.NE.0) GO TO 14
      IF (M.EQ.0) GO TO 19
C
C          TRANSFER CONTENTS OF JSTORE TO ISTORE
C
      DO 13 S=1,M
      ISTORE(S)=JSTORE(S)
   13 CONTINUE
C
      Z=M
      M=0
   14 I=ISTORE(Z)
      Z=Z-1
      GO TO 5
C
C          WE REACH HERE IF AN AUGMENTING PATH IS FOUND
C            ADJUST ALL FLOWS ALONG THIS PATH.
   15 Q=Q+1
      U=Q+LL
      IF (U.GE.RTEMP) GO TO 19
C          TEST LABEL ON VERTEX J
   16 I=NODE(J)
      IF (I.LT.0) GO TO 17
C
C     VERTEX IS WEAKLY LABELED. INCREMENT FORWARD FLOW BY 1
      FLOMAT(I,J)=1
      GO TO 18
C
C        VERTEX IS STRONGLY LABELED. DECREMENT BACKFLOW BY 1
   17 I=-I
      FLOMAT(J,I)=0
   18 J=I
C
C          IS SOURCE VERTEX REACHED?
      IF (J.EQ.SOURCE) GO TO 3
```

```
      GO TO 16
C
C           VERTEX LABELING AND FLOW AUGMENTATION IS COMPLETED
   19 IF (MARK.EQ.0) GO TO 20
      KMAT(SOURCE,SINK)=1
      KMAT(SINK,SOURCE)=1
   20 RETURN
      END
```

BIBLIOGRAPHY

1. Amin, A. T. and Hakimi, S. L., "Graphs with Given Connectivity and Independence Number or Networks with Given Measures of Vulnerability and Survivability," IEEE Trans. Circuit Theory, vol. CT-20, No. 1, p. 2-10, January 1973.

2. Baran, P., "On Distributed Communications Networks," IEEE Trans. on Communications Systems, p. 1-9, March 1964.

3. Boesch, F. T. and Frisch, I. T., "On the Smallest Disconnecting Set in a Graph," IEEE Trans. Circuit Theory, vol. CT-15, p. 286-288, September 1968.

4. Boesch, F. T. and Thomas, R. E., "On Graphs of Invulnerable Communications Nets," IEEE Trans. Circuit Theory, vol. CT-17, No. 2, p. 183-191, May 1970.

5. Chou, W. and Frank, H., "Survivable Communication Networks and the Terminal Capacity Matrix," IEEE Trans. Circuit Theory, vol. CT-17, No. 2, p. 192-197, May 1970.

6. Deo, N., "On Survivability of Communications Systems," IEEE Trans. Commun, Tech., p. 227-228, December 1964.

7. Ford, L. R. and Fulkerson, D. R., Flows in Networks, Princeton University Press, 1962.

8. Frank, H., "Vulnerability of Communication Networks," IEEE Trans. Commun. Tech., vol. COM-15, No. 6, p. 778-789, December 1967.

9. Frank, H. and Frisch, I. T., Communication, Transmission and Transportation Networks, Addison-Wesley, 1971.

10. Frank, H. and Frisch, I. T., "Analysis and Design of Survivable Networks," IEEE Trans. Commun. Tech., vol. COM-18, No. 5, p. 501-519, October 1970.

11. Frisch, I. T., "An Algorithm for Vertex-Pair Connectivity," Int. J. Contr., vol. 6, p. 579-593, 1967.

12. Frisch, I. T., "Flow Variation in Multiple Min-cut Calculations," J. Franklin Inst., vol. 287, p. 61-72, January 1969.

13. Gomory, R. E. and Hu, T. C., "Multiterminal Network Flows," J. SIAM, vol. 9, No. 4, p. 551-570, December 1961.

14. Kleitman, D. J., "Methods of Investigating Connectivity for Large Graphs," IEEE Trans. Circuit Theory, vol. CT-16, p. 232-233, May 1969.

15. Lin, S., "Computer Solutions of the Traveling Salesman Problem," Bell System Tech. Journal, vol. 44, p. 2245-2269, December 1965.

16. Lin, S. and Kernighan, B. W., "An Effective Algorithm for the Traveling Salesman Problem," Operations Research, vol. 21, No. 2, p. 498-516, March-April 1973.

17. Mayeda, W., "Terminal and Branch Capacity Matrices of a Communication Net," IRE Trans. Circuit Theory, p. 261-269, September 1960.

18. Steiglitz, K., Weiner, P., and Kleitman, D. J., "The Design of Minimum-Cost Survivable Networks," IEEE Trans. Circuit Theory, vol. CT-16, No. 4, p. 455- 460, November 1969.

INITIAL DISTRIBUTION

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Flag Officer In Command Headquarters Philippine Navy Roxas Blvd., Manila, Philippines	2
4.	Doctor Charles H. Rothauge Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
5.	Lt. Ruben F. Labre Headquarters Philippine Navy Roxas Blvd., Manila, Philippines	1

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Study of Deterministic Survivable Networks		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; December 1973 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ruben F. Labre		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December 1973 13. NUMBER OF PAGES 95
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The idea of survivability introduced as a network parameter has led to so many investigations. Several measures of survivability has been studied. The number of links and/or stations needed to be damaged to disrupt the system is the survivability criterion adapted in this study.

The development of analysis procedures for directed, undirected, or mixed networks based on the above criterion and use of the concepts in network flow and graph theory are treated in detail including computer program implementation

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |
(Page 1)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (cont'd)

of the algorithms. Finally a practical design algorithm for minimum-cost survivable network with respect to branch disconnection using a heuristic approach and analysis techniques is described.

Thesis
L13
c.1

147558

Labre
A study of determin-
istic survivable net-
works.